AFRL-IF-RS-TR-2002-10
Final Technical Report
February 2002

INTEGRATION OF NEXT-GENERATION INTRUSION DETECTION SYSTEM/EVENT MONITORING ENABLING RESPONSES TO ANOMALOUS LIVE DISTURBANCES (NIDES/EMERALD) INTRUSION DETECTION ENGINES WITH THE INTERNATIONAL OFFICE OF STANDARIZATION (ISO) ARCHITECTURE

SRI International

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J640/M522

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

APPROVED: Stanley E. Borek

Stanley E. Borek
Project Engineer

FOR THE DIRECTOR: [illegible]

WARREN H. DEBANY, Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE FEBRUARY 2002	3. REPORT TYPE AND DATES COVERED Final Apr 98 - Oct 01

4. TITLE AND SUBTITLE INTEGRATION OF NEXT-GENERATION INTRUSION DETECTION SYSTEM/EVENT MONITORING ENABLING RESPONSES TO ANOMALOUS LIVE DISTURBANCES (NIDES/EMERALD) INTRUSION DETECTION ENGINES WITH THE INTERNATIONAL OFFICE OF STANDARIZATION (ISO) ARCHITECTURE	5. FUNDING NUMBERS C - F30602-98-C-0059 PE - 62301E PR - F821 TA - 11 WU - 01
6. AUTHOR(S) Ulf Lindqvist and Phillip A. Porras	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) SRI International Computer Sciences Laboratory 333 Ravenswood Avenue Menlo Park California 94025-3493	8. PERFORMING ORGANIZATION REPORT NUMBER P01715-010
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington Virginia 22203-1714 Air Force Research Laboratory/IFGB 525 Brooks Road Rome New York 13441-4505	10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2002-10

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Stanley E. Borek/IFED/(315) 330-2095

12a. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes the expert-system-based intrusion detection technologies developed in the EMERALD program, and the research and experimentation performed with those components. The forward-reasoning expert-system tool P-BEST, which has been used to build signature-analysis engines for IDES, NIDES and now EMERALD, is described in detail. We show how data from network traffic interception, from host operating system audit trails, and from critical applications can be analyzed by P-BEST-based applications for real-time intrusion detection. The host-based and network-based intrusion detection monitors that we built have participated in various evaluations and experiments, confirming their detection capabilities and general applicability. We conclude that EMERALD's expert-system approach to misuse detection is well suited for the complex event analysis needed for wide attack coverage and near-zero false alarm rates.

14. SUBJECT TERMS Intrusion Detection System, Expert System, Computer Network, EMERALD, NIDES, IDS	15. NUMBER OF PAGES 112
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) environment is a distributed scalable tool suite for tracking malicious activity through and across large networks [46]. EMERALD introduces a highly distributed, building-block approach to network surveillance, attack isolation, and automated response.

1.1 EMERALD

EMERALD targets both external and internal threat agents that attempt to misuse system or network resources. It is an advanced highly software-engineered environment that combines signature-based and probabilistic analysis components with a higher-level aggregation and correlation units, all of which can be used iteratively and hierarchically. Its modules are designed to be independently useful, dynamically deployable, easily configurable, reusable, and broadly interoperable. Its design scales well to very large enterprises. The objectives include achieving innovative analytic abilities, rapid integration into current network environments, and much greater flexibility of surveillance whenever network configurations change.

EMERALD employs a building-block architectural strategy using independently tunable distributed surveillance monitors that can detect and respond to malicious activity on local targets, and can interoperate to form an analysis hierarchy. The basic architectural structure is shown in Figure 1.1. The figure shows analysis units surrounding the target-specific resource objects. It also shows the possible integration of third-party modules, including inputs derived from other sources, and outputs sent to other analysis platforms or administrators and emergency response centers. This architecture is explained in the following text.

A key aspect of this approach is the introduction of EMERALD monitors. An EMERALD monitor is dynamically deployed within an administrative domain to provide localized real-time analysis of infrastructure (e.g., routers or

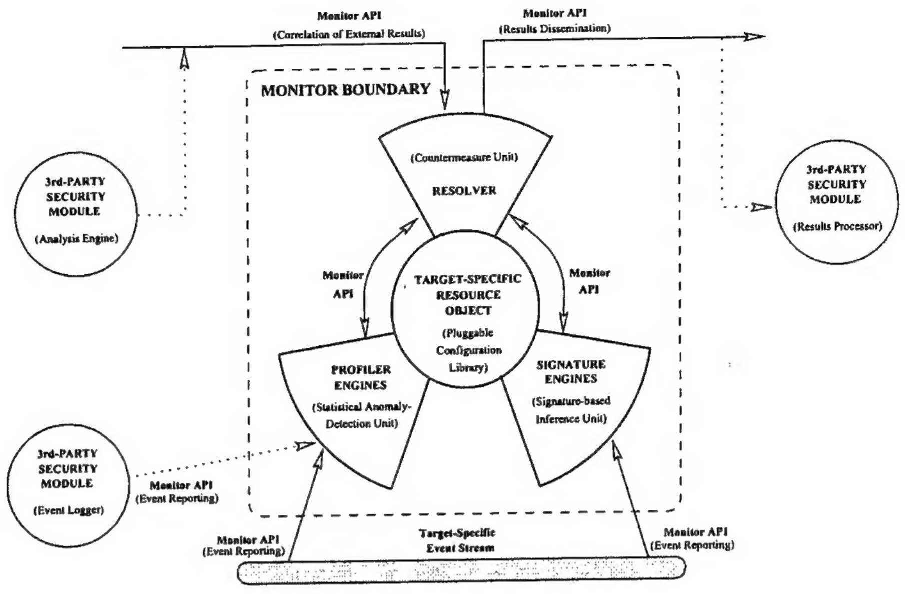


Figure 1.1: The Generic EMERALD Monitor Architecture

gateways) and service (privileged subsystems with network interfaces). An EMERALD monitor may interact with its environment passively (reading activity logs or network packets) or, potentially, actively (via probing that supplements normal event gathering [31]). As monitors produce analytical results, they are able to disseminate these results asynchronously to other client monitors. Client monitors may operate at the domain layer, correlating results from service-layer monitors, or at the enterprise layer, correlating results produced across domains.

The EMERALD framework supports the formation of a layered analysis hierarchy for recognition of more global threats to interdomain connectivity, including coordinated attempts to infiltrate or destroy connectivity across an enterprise.

Equally important, EMERALD does not require the adoption of this analysis hierarchy. Monitors themselves stand alone as self-contained analysis modules, with a well-defined interface for sharing and receiving event data and analytical results among other third-party security services. EMERALD's signature analysis subsystem employs a variant of the P-BEST (Production-Based Expert System Toolset) expert system that allows administrators to instantiate a rule set customized to detect known "problem activity" occurring on the analysis target. EMERALD also has a probabilistic analysis subsystem based on Bayesian techniques [59].

Fundamental to EMERALD's design is the abstraction of analysis semantics from the monitor's code base. Under the EMERALD monitor architecture, all analysis-target-specific information is contained within each resource object, specifying items from a pluggable configuration library. The resource object encapsulates all the analysis semantics necessary to instantiate a single service monitor, which can then be distributed to an appropriate observation point in the network. Resource-object elements customize the monitor for the analysis target, containing data and methods, such as the event collection methods, analytical module parameters, valid response methods, response policy, and subscription list of external modules with which the monitor exchanges alarm information. This enables a spectrum of configurations from lightweight distributed monitors to heavy-duty centralized analysis platforms.

In a given environment, service monitors may be independently distributed to analyze the activity of multiple network services (e.g., FTP, SMTP, HTTP) or network element (router, firewall). Resource objects are being developed for each analysis target. As each EMERALD monitor is deployed to its target, it is instantiated with an appropriate resource object (e.g., an FTP resource object for FTP monitoring, and a BSM resource object for BSM Solaris kernel analysis). The monitor code base itself is analysis target independent. As EMERALD monitors are redeployed from one target to another, the only thing that is modified is the content of the resource object.

Resource objects lend themselves to the key project objectives of reusability and fast integration to new environments. The project is developing a library populated with resource objects that have been built to analyze various service and network elements. Installers of EMERALD will be given our monitor code base, which they do not have to touch. They can then download appropriate resource objects associated with their analysis targets, modify them as desired, and instantiate the monitors with the downloaded resource objects.

The project is also working toward new techniques in alarm correlation and management of analytic services. The concept of composable surveillance will allow EMERALD to aggregate analyses from independent monitors in an effort to isolate commonalities or trends in alarm sequences that may indicate a more global threat. Such aggregate analyses are classified under four general categories: commonality detection, multiperspective reinforcement, alarm interrelationships, and sequential trends.

Briefly, commonality detection involves the search for common alarm indicators produced across independent event analyses. In such cases, the results from one monitor's analyses may occur under a threshold that warrants individual response, but in combination with results from other monitors may warrant a global response. This approach can address low-rate distributed attacks and cooperative attacks, as well as widespread contamination effects. Multiperspective analysis refers to efforts to independently analyze the same target from multiple perspectives (e.g., an analysis of a Web server's audit logs in conjunction with Web network traffic). Alarm interrelationships refer to EMERALD's ability to have a monitor model an interrelationship (cause and effect) between the occurrence of alarms across independent analysis targets. For example, an alarm regarding activity observed on one host or domain may give rise to a warning indicator for a different threat against a second host or domain. Last, sequential trends in alarms seek to detect patterns in alarms raised within or across domains. These patterns of aggressive activity may warrant a more global response to counteract than can be achieved by a local service monitor.

The EMERALD project represents an effort to combine research from distributed high-volume event correlation with over a decade of intrusion detection research and engineering experience. It represents a comprehensive attempt to develop an architecture that inherits well-developed analytical techniques for detecting intrusions, and casts them in a framework that is highly reusable, interoperable, and scalable in large network infrastructures. Its inherent generality and flexibility in terms of what is being monitored and how the analytical tools can be customized for the task suggest that EMERALD can be readily extended for monitoring other forms of malicious and nonmalicious "problem activities" within a variety of closed and networked environments.

1.2 Experience gained

This section summarizes our experience in the EMERALD development thus far.

1.2.1 Earlier experience

EMERALD has drawn on our earlier experience in developing and using IDES (Intrusion Detection Expert System [37]) and its successor NIDES (Next-Generation IDES [3]. Particularly for those people who are not aware of our earlier work, we summarize a few conclusions.

- From IDES, we attained considerable flexibility and runtime efficiency in the use of P-BEST [37]), which we have now adapted into EMERALD's pluggable analysis-engine framework as a self-sufficient component. The P-BEST approach proved to be very useful, and rules are relatively easy to write. P-BEST was adapted by Alan Whitehurst from its previous incarnation in MIDAS [53]. IDES also gave us the second generation of our statistical algorithms [27], begun in 1983 in an earlier project [28].

- From the NIDES development [3], several observations influenced the EMERALD effort. (1) Much of the available audit data (e.g., from C2 Unix and BSM) was not naturally well suited for our analytical purposes, and different sources of data would have been desirable. Greater abstraction would have been useful. (2) Although we did experiment with some higher-level audit data (from database management systems in relatively closed environments), attempting to detect misuse was less fruitful because the security policies of the DBMSs generally permitted what was closer to acceptable behavior. (3) We recognized that the NIDES statistical detection system as then configured would not scale well to distributed and networked environments, for two reasons. First, the measures needed to be treated in their entirety, rather than subsetted – as would be desirable for lightweight instances. Second, the results were not in a form that could be used recursively at a higher-layer instance. (4) We recognized the importance of the administrator interface, and observed that its complexities are unavoidable if flexibility in detection and response is required. However, we initially spent too much effort on developing our own GUI tools, until we decided to rely on some newly developed generic tools. In retrospect, we believe we would have progressed faster if we had had more emphasis on software engineering and on in-house applications.

- From the NIDES Safeguard effort [4], we observed that profiling functionality proved to be more effective than profiling individual users. That approach resulted in far fewer profiles, each of which tended to be much more stable. The resulting false-positive and false-negative rates were reduced considerably. We concluded that statistical analyses could be very effective in dealing with systems and subsystems such as servers and routers. (As a consequence, EMERALD subsequently broadened the statistics algorithms to improve handling of network protocols, by having a master profile of client usage against which a single service can be compared. For example, anonymous FTP sessions can simultaneously be profiled against the master profile for anonymous sessions.)

These observations have had a significant impact on the EMERALD architecture and its implementation, particularly in moving to a distributed and networked target environment.

1.2.2 EMERALD experience

The underlying generic analysis-engine infrastructure uniformly wraps the signature analysis, probabilistic engine, and any future engines we might wish to integrate. The infrastructure provides the common EMERALD API, event-queue management, error-reporting services, secondary storage management, and internal configuration control. The infrastructure was assembled first for the EMERALD statistics component (estat), but proved its generality when we attempt to integrate P-BEST as the EMERALD expert system (eXpert). The integration of P-BEST inference engines required some linkage code to bind with the underlying EMERALD libraries, and is now automatically generated as part of the compilation process.

After more than two years developing EMERALD, our experience thus far is summarized as follows.

- Generality of approach. We have attempted to solve some difficult problems rather generally, and have typically avoided optimizing our approach to any domain-specific assumptions. In particular, the decoupling of generic and target-specific concepts simplifies reusability of components and extensibility, and enhances integration with other data sources, analysis engines, and response capabilities. The hierarchically iterative nature permits analyses with broader scope across networks and distributed systems. Although the advantages of such a farsighted approach may not be evident until EMERALD is more widely used and extended to new application areas, we firmly believe that this approach can be very instructive to us and to other groups, from the perspective of research and development potential – and can have major long-term advantages. (Platform-specific optimizations are of course possible, if they are deemed necessary.)

- Software engineering. We believe that our strong emphasis on good software engineering practice in EMERALD has already had substantial payoffs, particularly in enabling us to rapidly incorporate different analytic engines into the generic framework. (The modularization and integration of the P-BEST expert system component is discussed below.) This emphasis clearly improves the general evolvability of the system, and also has significant benefits with respect to interoperability – within EMERALD, with independently developed analysis engines, with analysis data from arbitrary sources, and in terms of the distribution of analysis results. The software-engineering emphasis also helps facilitate the iterative use of EMERALD analytic engines by making the layered instances of the system symmetric.

- Scope of applicability. We believe that our attention to software engineering simplifies the broadening of EMERALD's domains of applicability – for example, detecting, analyzing, and responding to potential threats to survivability, reliability, fault tolerance, and network management stability. There is nothing intrinsic in the EMERALD architecture and implementation that would limit its applicability. The application to requirements other than security is basically a matter of writing or modifying the relevant resource objects and configuring the system appropriately, and is not expected to require major changes to the existing analysis infrastructure.

- Relative merits of various paradigms. It should be no surprise to those in the intrusion detection community that signature-based analysis is good at detecting and identifying well-defined known scenarios, but very limited in detecting hitherto unknown attacks (except for those that happen to trigger existing rules serendipitously). On the other hand, statistical profile-based analysis can be effective in detecting unknown attacks and providing early warnings on strangely deviant behaviors; however, the statistical approach does not naturally contribute to an automated identification and diagnosis of the nature of an attack or other type of deviation that it has never identified before. Although inferences can be drawn about the nature of an anomaly, based on the statistical measures that were triggered, further reasoning is typically necessary to identify the nature of the anomaly – for example, is it an attack in progress, or a serious threat to system survivability.

  Precisely because it is aimed at detecting potentially unforeseen threats rather than very specific scenarios that can be easily detected by signature-based analyses, the statistical component can be expected to turn up false positives. In the EMERALD framework, this is not necessarily a problem. We believe it is much more effective for the resolver to discard statistical anomalies that it deems nonserious rather than try to reduce the false positives in the statistical component itself (which requires greater knowledge of the potential threats – which is what can otherwise be avoided). Furthermore, once new attacks and threats are identified, it is desirable to add new rules to the expert-system rule base.

Overall, we believe that each type of analysis (such as the expert system, the statistical component, the resolver, or any additional analysis engines) will have its own areas of greatest effectiveness, but that no one paradigm can cover all types of threat. Therefore we endorse a pluralistic approach. Inference and reasoning engines, Bayesian analysis, and other paradigms may also be applicable to detection, identification, and resolution of the nature of anomalies and attacks.

- Local, hierarchical, and distributed correlation. One of the most far-reaching observations relates to the importance of being able to correlate local results from different target platforms at the same or different layers of abstraction, and also to correlate results relating to different aspects of system behavior. The inherent layered iterative nature of the EMERALD architecture is significant in this respect, because the same analytic component can be used at different layers of abstraction. We are just now beginning to conduct some experiments to demonstrate the power of this approach. In so doing, we are extending the existing EMERALD resolver to interpret the results of different analytic engines and to recommend responses appropriate to the specific layer of abstraction. Further analytic engines may also be required at various layers of abstraction, such as some reasoning tools.

- Importance of further research, prototype development, and experimentation. EMERALD continues to explore advanced concepts, as did IDES and NIDES. Although most of the necessary analysis infrastructure is now in place, R&D advances are still required for EMERALD relating to inference necessary to enhance correlation in the analysis of and response to coordinated attacks and interdependent anomalies in distributed environments, and in generalizations of applicability beyond security. These are ongoing efforts.

- Interoperability. The Common Intrusion Detection Framework (CIDF) and the ongoing IETF standardization effort are important. Both are expected to increase the interoperability within and among different analysis and response systems. EMERALD is very much in line with these efforts, and compatibility is not expected to be a problem. CIDF interface definitions are based on an architectural decomposition that is aligned closely to that of EMERALD's monitor design. In particular, EMERALD's target-specific event-generation components are equivalent in function to CIDF

E-boxes; EMERALD's statistical and signature analysis engines are equivalent in function to CIDF A-boxes; EMERALD's resolver is equivalent in function to a CIDF R-box. In hierarchical composition, an EMERALD service layer monitor is capable of passing alerts to a domain monitor. The service layer monitor can operate as a CIDF E-box, and the domain monitor can operate as a CIDF A-box. CIDF working documents are available online (*http://www.isi.edu/~brian/cidf/*).

1.2.3 EMERALD's expert system

With respect specifically to the integration of P-BEST into EMERALD [32], our experience has strongly reinforced our conceptual framework.

- The software engineering quality of the EMERALD monitor architecture was put to a test when a summer visitor previously unfamiliar with the system joined us to integrate the signature analysis engine into the generic monitor framework. The statistical anomaly detection engine had been developed in concert with the EMERALD API, and the NIDES expert-system-based signature engine was the first additional component to use the API. The revision and integration procedure went very rapidly (about a man-week), and minor problems that were discovered and solved were due to constraints in the expert-system tool rather than in the EMERALD API. This supports our claim that the EMERALD API is well suited for integration of various kinds of third-party modules into the monitor architecture. Although this is not an exciting *gotcha,* it was important to the development effort.

- The data-driven nature of the EMERALD monitors makes the intermonitor and intramonitor message passing a central function of the API. The programmer is provided with a set of abstract data types, including a set of methods to handle messages and fields within messages. An example of a powerful feature of the EMERALD message format is the possibility of defining a message field as an array of message fields. This allows the programmer to effectively encapsulate one EMERALD message inside another. In the signature-analysis engine, this capability is used to include the original event record(s) in every alert message sent to the resolver, in addition to the information provided by the triggered rules. This also allows a hierarchy of analysis units (including resolvers) to be able to pass along any or all information produced earlier.

- The generality of the API with respect to the abstract data types is also reflected by the ease with which we were able to write a code-generation utility for the interface code that connects the expert system to the monitor. This utility is used when redirecting the signature-analysis engine to a completely new event stream, using the information in the resource object to fit the engine to the analysis target. The purpose of the utility is to relieve the creator of a resource object from the inner workings of the monitor. The API design made it easy to isolate the target-dependent code and let it be machine generated.

1.3 Scope of this report

This report covers many, but not all aspects of the EMERALD research and development program. The project, for which this report serves as a final technical report, was focused on development of expert-system-based intrusion detection technology, and integration with other Information Assurance & Survivability technologies. Other EMERALD activities, such as probabilistic detection and correlation technologies, mission-based alert correlation, correlated attack modeling, and attack response, are described in the project reports pertaining to their specific projects, respectively.

Chapter 2

An Expert System for Intrusion Detection

This chapter describes P-BEST [32], the expert system tool that is the basis of all rule-based intrusion detection monitors in EMERALD. In addition to the present project, the work on P-BEST was supported by DARPA/AFRL under contract number F30602-96-C-0294.

2.1 Introduction

Intrusion detection components analyze system and user operations in computer and network systems in search of activity considered undesirable from a security perspective. Data sources for intrusion detection may include audit trails produced by an operating system, or network traffic flowing between systems, or application logs, or data collected from system probes (e.g., file system alteration monitors). The collected data may be stored for batch-mode analysis or immediately analyzed in real time.

For the most part, the various strategies for intrusion detection are not unique to the field, but are rather derived from applications established by other fields: knowledge-based expert systems, pattern recognition algorithms, statistical profiling techniques, neural networks, Bayesian statistics, information retrieval algorithms, state-transition models, Petri-net techniques, and so forth. Among the more widely used strategies proposed early within the intrusion detection community are signature-based analyses.

Intuitively, we describe a signature-based intrusion-detection component as an algorithm with which we specify the characteristics of malicious behavior and then monitor an event stream for activity that maps to the target behavior. Various signature-based systems have been developed, ranging from simple (but efficient) pattern-matching systems to more sophisticated algorithms that employ more general directed reasoning systems such as rule-based expert systems. In this chapter, we describe in detail the principles and language of one forward-chaining rule-based expert system construction toolset called P-BEST (Production-Based Expert System Toolset), which has been continually applied to intrusion detection applications for more than a decade, but never before widely presented in this level of detail.

By using a general expert system, we can describe the behavior of our signature-based intrusion detection component within an established theoretical framework. This choice also facilitates the evolution of the component, because new rules can be added without changing existing rules and without creating any undesired dependency. Traditional reasons for not choosing an expert system are related to low performance, difficult integration with other program components, and language complexity. However, in this report we show that P-BEST is sufficiently fast for real-time detection of currently widely used attack methods—SYN flooding and buffer overruns—against which systems usually have no defense mechanisms. We also show that P-BEST provides exceptional interoperability with native operating system libraries, and is easily integrated into a larger software framework for distributed anomaly and misuse detection. We also argue that while the production rule language is powerful, it remains easy to use for beginners.

2.2 Monitoring misuse through expert systems

Expert systems provide strategies and mechanisms for processing facts regarding the state of a given environment, and deriving logical inferences from these facts. With respect to intrusion detection, a fact maps to an event that is recorded and evaluated by the expert system. This process of fact evaluation leading to the assertion of a new derived fact or conclusion is referred to as *modus ponens*, which states that given $(p \Rightarrow q)$ and p we deduce q. Systems that iteratively apply modus ponens under a bottom-up reasoning strategy (from evidence evaluation to conclusion) are referred to as *forward-chaining* systems. Forward-chaining expert systems are well suited for reasoning about activity within an event stream. A forward-chaining rule-based system is data driven: each fact asserted may satisfy the conditions under which new facts or conclusions are derived. Alternatively, *backward-chaining* systems employ the reverse strategy; starting from a proposed hypothesis they proceed to collect supportive evidence. Backward-chaining systems are typically applied to problems of diagnosis, whereas forward-chaining strategies dominate systems involving prognosis, monitoring, and control applications.

Using a forward-chaining rule-based system, one may establish a chain of rules, or *rule set*, with which a series of asserted facts may lead the system to deduce that a targeted multistep scenario has occurred. Within an intrusion detection system, event records are asserted as facts and evaluated against penetration rule sets. As individual rules are evaluated against facts and satisfied, the individual event records provide a trail of reasoning that allows the user to analyze the evidence of malicious activity in isolation from the full event stream. In this section, we discuss the basic elements of forward-chaining rule-based systems, and provide an overview of the P-BEST expert system and its language.

2.2.1 Components of forward-chaining systems

The underlying strategy of a forward-chaining reasoning system involves the atomic evaluation of each fact presented to the system against conditional expressions that, when satisfied by the arguments of a fact, establish new derived facts or conclusions. In this context, a *fact* is a statement that is asserted into the system and whose validity is accepted (for example, "smoke is present"). Facts are often implemented as attributes and values that represent the state of the environment to which the expert system is applied. A *rule* is an inference formula of the form $\phi_1, \ldots, \phi_n$ *infer* ψ. Inference formulae can be alternatively expressed as *production rules*, such as `IF` ... `THEN` .... Production rules are the basic elements through which an expert system is programmed to interpret and discover meaning from environmental signals that it receives, as in

`IF` *smoke is present* `THEN` *fire is near.*

A production rule consists of two parts, the *antecedent* (or conditional part, left-hand side) and the *consequent* (or right-hand side) as shown in Figure 2.1. When the *conditions* (predicate expressions) in the antecedent are satisfied, the rule is *activated.* The logical component through which an expert system evaluates a fact against the production rules is referred to as the *inference engine.* As an antecedent is found to be satisfied by the attributes of a fact, the consequent of the rule is asserted to hold, and the rule is said to have *fired.* Expert systems might additionally allow the inference engine to initiate action within the consequent, for example:

`IF` *fire is near* `THEN` **initiate** *sprinkler.*

Abstractly, the assertion of action, such as the initiation of a response, based on a fact derived from an inference engine is placed within the purview of a *decision engine*, though in practice inference and response may be merged.

$$\boxed{\begin{array}{l} \texttt{IF} \\ \quad\left.\begin{array}{l} condition_1 \\ condition_2 \\ \quad\vdots \end{array}\right\} \text{antecedent} \\ \texttt{THEN} \\ \quad\left.\begin{array}{l} derived\_fact_1 \\ derived\_fact_2 \\ \quad\vdots \end{array}\right\} \text{consequent} \end{array}}$$

Figure 2.1: Production rule structure.

The collection of facts available to the system at any point in time is called the *factbase* (or working memory) of the system. The collection of rules is called the *knowledge base* (or production memory). Although separation of data (facts) from knowledge (rules) is an important abstraction within rule-based expert systems, some texts use the terms more loosely and consider the factbase to be part of the knowledge base. Another important abstraction is the separation of knowledge from the inference engine. In practice, an inference engine, also known as an expert system *shell*, provides several advantages over a one-of-a-kind system written in a procedural language. In particular, a knowledge-independent shell can be used to develop expert systems for many different knowledge domains. The knowledge in the expert system can also be incrementally extended by adding new rules, as opposed to implementing large portions of the decision process all at once. Next, we present the principles and language of P-BEST, a construction toolset for building customized inference engines, and discuss its applicability to intrusion detection.

2.2.2 An overview of P-BEST

The Production-Based Expert System Toolset (P-BEST) was originally written by Alan Whitehurst, and employed in the Multics Intrusion Detection and Alerting System (MIDAS) [53], which performed misuse detection on the National Computer Security Center's Internet-connected mainframe, Dockmaster. P-BEST was later enhanced at SRI by Whitehurst, and later by Fred Gilham, and was employed in an early version of the Intrusion Detection Expert System (IDES) [36], and later Next-Generation IDES (NIDES) [3]. See Section 2.3 for details on the application of P-BEST on these systems.

The P-BEST toolset consists of a rule translator, a library of runtime routines, and a set of garbage collection routines. When using P-BEST, rules and facts are written in the P-BEST production rule specification language. The rule translator, *pbcc*, is then used to translate the specification into a C language expert-system program. This expert system can then be compiled into either of two forms: a stand-alone self-contained executable program or a set of library routines that implement the core P-BEST inference engine, and which can be linked to a larger software framework. P-BEST has several features that make it well suited for the type of application described in this report:

- The P-BEST language is small and relatively intuitive to use and extend.
- It is easily applied to a variety of problem domains. P-BEST provides a general-purpose forward-chaining inference engine that can be targeted to a specific application domain. P-BEST does not inherently depend on the structure of the input data stream or the inference objectives of the application that employs it.
- By using translation instead of interpretation of rules, P-BEST can be used to build expert systems for performance-demanding applications.

A pre-compiled expert system, rather than an expert-system interpreter, provides a significant advantage in performing real-time event analysis.

- Pre-compilation also allows P-BEST components to be integrated well into larger program frameworks, as they are easily called from, and can call out to, other C libraries. Arbitrary C functions can be called from the antecedent or consequent of any P-BEST rule. Thus, it is possible to write powerful rules without adding unnecessary complexity to the P-BEST language.

2.2.3 The P-BEST language

P-BEST provides a production rule language from which users may specify the inference formula for reasoning and acting upon facts asserted into its factbase from external sources or derived from the satisfaction of other production rules. This section provides a brief overview of the principal elements of this language, with common examples of its usage. The language overview provides a primer for understanding several examples of intrusion detection rules later in this chapter.

In P-BEST, the structure of a fact is specified by the user through a template definition referred to as a pattern type or *ptype*. For example, to define a ptype named *event* that consists of the four fields *event_type* (an integer), *return_code* (an integer), *username* (a string), and *hostname* (a string), we define the fact template as in Figure 2.2. Facts from such a ptype definition could be constructed through the monitoring of audit records and asserted into the factbase for evaluation against the available production rules.

```
ptype[event event_type:int,
            return_code:int,
            username:string,
            hostname:string]
```

Figure 2.2: An example of a ptype declaration.

Fact evaluation is performed by the P-BEST inference engine, where the attributes of the fact are mapped against the predicate expression(s) of each rule antecedent. For example, we may want to determine whether the asserted fact represents an unsuccessful login attempt, which we shall refer to as *e*. To express this criterion using a mathematical notation style, we can form the statement in Equation 2.1.

Here, S represents the set of all facts known to the P-BEST factbase, and within which a production rule antecedent postulates the existence of a fact

$$\Big(\exists e\Big)\Big((e \in S) \land \mathrm{event}(e) \land (e_{event\_type} = login) \land (e_{return\_code} = bad\_password)\Big) \qquad (2.1)$$

e that satisfies specific properties. In the P-BEST language, the statement in Equation 2.1 placed in the antecedent of a rule would be written as in Figure 2.3.

```
[+e:event|event_type == login,
          return_code == BAD_PASSWORD]
```

Figure 2.3: An example of fact matching.

The term `e:event` allows one to assign an *alias* `e` to one fact (of possibly several) that satisfies the antecedent for the duration of the rule. The plus (+) sign after the opening bracket represents an existential quantifier that allows the rule to check for any fact that satisfies the conditions of the antecedent. Alternatively, a minus (-) sign searches for cases where no fact in the factbase satisfies the conditions of the antecedent. For example,

```
[-event|username == "GoodGuy"]
```

evaluates to true if there is no event in the factbase that has been asserted on behalf of "GoodGuy."

The plus and minus tests have corresponding assert and delete actions that can appear in the consequent of a rule. For example, to assert a new fact of ptype `bad_login` and give its fields initial values, we can write

```
[+bad_login|username = e.username, hostname = e.hostname]
```

To be deleted from the factbase, a fact must be matched and given an alias in the antecedent before it can be deleted in the consequent. This is illustrated in the example of a complete rule named `Bad_Login` in Figure 2.4.

```
rule[Bad_Login(#10;*):
   [+e:event| event_type == login,
              return_code == BAD_PASSWORD]
==>
   [+bad_login| username = e.username,
                hostname = e.hostname]
   [-|e]
   [!|printf("Bad login for user %s from \
      host %s\n", e.username, e.hostname)]
]
```

Figure 2.4: An example of a rule declaration.

The `Bad_Login` rule in Figure 2.4 also demonstrates how the evaluation of an asserted fact can be used to derive subsequent facts that may themselves drive new inferences. That is, in the above rule, should a login event be encountered with a return code of `BAD_PASSWORD`, the rule creates a new fact of ptype `bad_login`, which saves the username and hostname of the event; the rule also destroys the event fact `e` from the factbase. Using a mathematical notation, we can represent this state transition in our factbase from S to a desired new state

S' as in Equation 2.2 (this excludes lines 8 and 9 in Figure 2.4).

$$\frac{\Big(\exists e\Big)\Big((e \in S) \wedge \text{event}(e) \wedge (e_{event\_type} = login) \wedge (e_{return\_code} = bad\_password)\Big)}{\vdash \Big(S' = S - \{e\} \bigcup \{\text{bad\_login}(b) \mid (b_u = e_u) \wedge (b_h = e_h)\}\Big)} \tag{2.2}$$

Within parentheses after the rule name (line 1), there is a semicolon-separated list of options. The option #10 means that this rule is given a ranking (priority) of 10. Priorities allow one to specify well-defined orders in the sequences for rule evaluation, and are primarily used for rules required to be evaluated first for initialization purposes, or that must be evaluated last to perform garbage collection. The star option (*) indicates that the rule is repeatable, that is, the rule is allowed to fire repeatedly even if no other rule is fired in between. Thus, a key function of the consequent is to alter the state of the factbase such that the antecedent is not satisfied indefinitely (e.g., the consequent may mark or remove a fact). The arrow delimiter (==>) separates the antecedent and the consequent (line 4).

The `[!|...]` clause (line 8) within the consequent illustrates how the P-BEST inference engine may call out to native C functions should action be warranted when the antecedent is evaluated to true. Both inference and action can be taken directly within the P-BEST inference engine. P-BEST recognizes most of the standard library C functions, which may be invoked directly via the `[!|...]` clause, and which may refer to ptype attributes directly. User-defined C functions and auxiliary variables may also be invoked and referenced, respectively. To do this, we must declare our intentions to reference C variables and functions using the P-BEST external type declaration mechanism *xtype*. For example, the following external declarations will allow P-BEST to recognize a user-defined C function called *native_probe()* returning an integer and an integer variable *end_of_stream* as follows:

```
xtype [native_probe: intfunc]
xtype [end_of_stream: int]
```

We can then employ our native C routine and variable directly in a P-BEST production rule, as illustrated in Figure 2.5. The antecedent `[?|...]` clause (line 3) is a query clause used to evaluate conditional requirements. This rule will check to see whether the `end_of_stream` variable has been set to 1, and if not, it will set the variable to the return code of the function `native_probe()` (line 5), which is invoked in the consequent. This *native_probe()* could, for example, provide an interface to the host operating system that allows the expert system to retrieve application records, which it may then assert as facts in the factbase. The rule also gives an example (line 6) of how a field in an existing fact can be modified; in this case, the field `rec_cnt` of the fact `counter`, aliased in the antecedent, is incremented by 1.

To further improve the performance of the expert system, rules can be disabled and enabled dynamically through actions in the consequents of rules. A rule can even disable itself, which means that it can fire once, at most, unless

```
rule[get_native_record(-99;*):
    [+c:counter]
    [?|'end_of_stream != 1]
==>
    [!|'end_of_stream = native_probe()]
    [/c|rec_cnt += 1]
]
```

Figure 2.5: Example usage of external C types.

enabled again by another rule. To disable a rule, we can put the following action in a consequent:

```
[-#rulename]
```

To enable a rule, we can change the minus sign in the above statement to a plus sign. In addition, a rule can be declared as disabled from start by adding a single minus sign to the list of options after the rule name, for example:

```
rule[rulename(#10;*;-):
```

Using these features, we can build preconditional requirements that can enable or disable whole portions of the knowledge base, depending on the current state of the environment being monitored. For example, rules pertaining to the analysis of a service *A* can be dynamically added or removed from the knowledge base by the expert system itself, depending on whether service *A* is currently enabled or disabled within the analysis target. Another example is when the analysis is extended with previously disabled rules due to an increased level of suspicion reported by the basic rule sets.

Another powerful feature of P-BEST is the ability of rules to uniquely mark and unmark facts, and to test for these marks. This can be used when we want to give several groups of mutually exclusive rules the chance to examine a fact before it is deleted from the factbase. Each rule will evaluate the fact, and if the antecedent is satisfied, the consequent of the rule will mark the fact. This will allow the rule to avoid re-firing, while not having to remove the fact completely from the factbase. When all such rules have evaluated (and if necessary marked) the fact, the fact can then be removed by a lower-priority fact-removal rule that is run last. For example, to match an event that is not marked with CHECKED, we can put the following test in the antecedent of our rule:

```
[+e:event^CHECKED]
```

To mark a matched event fact e with CHECKED, we can add the following action to the consequent:

```
[$|e:CHECKED]
```

Alternatively, to unmark a fact we simply use a caret (^) instead of the dollar sign ($):

```
[^|e:CHECKED]
```

Finally, we can use the dollar sign to check for a marked fact, as follows:

```
[+e:event$CHECKED]
```

2.2.4 P-BEST language simplicity and usability tested in student experiment

Although the P-BEST language has proven itself suitable for intrusion detection systems, it is in fact also a general language for building rule-based expert systems in many different applications. The close integration with C makes it unnecessary to include more than the basic operations in the P-BEST language itself, because any needed operation can be designed as a C function and called from the antecedent or consequent of a P-BEST rule. Thus, the P-BEST language can be kept small and simple, resulting in a very low learning threshold for beginners.

In addition to its use in intrusion detection system development, P-BEST has recently for the first time been used for laboratory exercises in a university course in applied computer security at Chalmers. In addition to the educational goals of these exercises, we wanted to learn what amount of instruction is required for beginners when applying P-BEST to intrusion detection analysis and thereby see whether the experiment would support or contradict our hypothesis that P-BEST is easy to use for beginners.

The assignment was to build a system that could be used to automatically detect attacks against a file transfer (FTP) server. For evaluation of their resulting system, the students were given a very large data file (3 megabytes of text) containing recorded network data representing actual FTP transactions. A small number of real and synthetic intrusions were mixed with a large number of normal transactions, and the students were to use their system to find those intrusions. It was supposed to be a pedagogic effect that the file was too large to be easily examined by hand, because this is the very reason for having automatic intrusion detection tools. It was also required by the students to include in their lab reports a discussion of their experiences of using the tool.

There were 87 students who participated in the assignment, and with a few exceptions they worked in pairs, making a total of 46 groups. The estimated maximum working time was two lab sessions of 4 hours each, plus another 8 hours of homework to prepare the lab sessions and to complete the report. Out of the 46 groups, 25 had built a system that gave the completely correct answer. An additional eight groups would most likely have got the correct result if they had not all misinterpreted a vaguely formulated part of the instructions. Only a handful of groups failed to hand in a report before the given deadline. Most students reported that they found the exercise interesting, and some even took the time to give detailed suggestions of improvements to the tool. As we had expected, being used to writing programs in a procedural style, they had some initial difficulties in declarative programming. In summary, we claim that the student experiment shows that P-BEST has a low learning threshold for beginners and is thereby suitable both for building user-customizable intrusion detection systems and for student exercises in computer security courses.

2.3 Integration of P-BEST into IDS components

For more than 10 years, P-BEST has been successfully integrated into several intrusion detection systems (IDSs) that represent the state of the art for their time. The application of P-BEST to intrusion detection began in the mainframe world of Multics and lands in present time with the highly distributed, scalable, and network-oriented EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) environment. It is not only the IDSs that have changed over time; P-BEST itself has been continuously improved as the requirements and its operational environment have changed. However, performance and language simplicity are issues that have had top priority from the beginning, and are no less important today.

2.3.1 P-BEST in MIDAS

P-BEST was developed at SRI International and first deployed as the core of MIDAS, which provided real-time intrusion and misuse detection for the National Computer Security Center's networked mainframe, Dockmaster, a Honeywell DPS-8/70 running Multics [53]. Audit data preprocessing and command monitoring was performed on the Dockmaster, and the data was sent to a separate Symbolics Lisp machine where the expert system and the user interface were running.

MIDAS used both static and dynamic knowledge for detecting intrusive user behavior. The static knowledge was represented in so-called immediate attack heuristics written as P-BEST rules that would trigger on events that were considered anomalous regardless of previous system activity. In terms of dynamic knowledge, MIDAS recorded user and system statistics in a database that would represent normal behavior. It is interesting to note that it was in fact another set of P-BEST rules—the user anomaly heuristics and the system state heuristics—that used threshold values derived from the statistics database to distinguish anomalous user and system behavior from normal activity. Thus, the P-BEST inference engine was the sole analysis component in MIDAS.

2.3.2 P-BEST in IDES and NIDES

In 1983, SRI International began research on statistical techniques for audit-trail reduction and analysis [17]. This research led to the development of a prototype IDES, capable of providing real-time detection of security violations on single-target host systems. Originally, IDES used only statistical anomaly detection [16, 27], but later a component for misuse detection based on static knowledge was added, using P-BEST [36]. The two components were fed the same audit records, but performed their inferences and reporting independently.

Next, SRI began a comprehensive effort to enhance, optimize, and re-engineer the earlier IDES prototype into a production-quality intrusion detection system with the name Next-Generation Intrusion Detection Expert System (NIDES). Just like its predecessor, NIDES has both a statistical anomaly detection component and a rule-based misuse detection component [3]. Again, P-BEST was the expert system shell of choice for the rule-based component, but P-BEST was first extensively revised. Among other things, the revision gave P-BEST a new syntax and a very tight coupling to the C programming language. While the early version of P-BEST used in MIDAS and IDES compiled rules into Lisp object code, the new version produced C source code. NIDES collects host audit trail data from different host systems and converts it to the NIDES audit record format. The current version of NIDES has a default rulebase of 39 rule sets (69 total production rules) but also allows the user to write his or her own rules (that, for example, are specific to the user's environment or policy) and has a mechanism for dynamically adding new rules at runtime.

2.3.3 P-BEST in the EMERALD eXpert

The EMERALD environment is a distributed scalable tool suite for tracking malicious activity through and across large networks [46]. EMERALD employs a building-block architectural strategy using independent distributed surveillance *monitors* that can analyze and respond to malicious activity on local targets, and can interoperate to form an analysis hierarchy. The generic EMERALD monitor architecture is designed to enable the flexible introduction and deletion of analysis engines from the monitor boundary as necessary. In its dual-analysis configuration, an EMERALD monitor instantiation combines signature analysis with statistical profiling to provide complementary forms of analysis over the operation of network services and infrastructure. In general, a monitor may include additional analysis engines that can implement other forms of event analysis, or a monitor may consist of only a single resolver implementing a response policy based on intrusion summaries produced by other EMERALD monitors. Monitors also incorporate a versatile API that enhances their ability to interoperate with the analysis target, and with other third-party intrusion detection tools.

Underlying the deployment of an EMERALD monitor is the selection of a target-specific event stream. The event stream may be derived from a variety of sources, including audit data, network datagrams, SNMP traffic, application logs, and analysis results from other intrusion detection instrumentation. The event stream is parsed, filtered, and formatted by the target-specific event-collection methods provided by the monitor's pluggable configuration library, referred to as the *resource object*. Event records are then forwarded to the monitor's analysis engine(s) for processing.

The EMERALD *eXpert* (pronounced E-expert) is a generic signature-analysis engine based on the expert system shell P-BEST. The eXpert resource object has two parts, one of which consists of the configuration files for the EMERALD API that define the transports used for message passing (e.g., files or network connections), the message templates, and so forth, for the particular analysis target. The other part of the resource object is a P-BEST source file containing the fact type (ptype) declarations and rules. In the ptype declarations, the user must specify to what message field (if any) the ptype field corresponds.

Under EMERALD's eXpert architecture, special-purpose rule sets are encapsulated within resource objects that are then instantiated with an EMERALD monitor, and which can then be distributed to an appropriate observation point in the computing environment. This enables a spectrum of configurations from light-weight distributed eXpert signature engines to heavy-duty centralized host-layer eXpert engines, such as those constructed for use in NIDES and MIDAS. In a given environment, P-BEST-based monitors may be independently distributed to analyze the activity of multiple network services (e.g., FTP, SMTP, HTTP) or network elements (e.g., a router or firewall). As each EMERALD eXpert is deployed to its target, it is instantiated with an appropriate resource object (e.g., an FTP resource object for FTP monitoring), while the eXpert code base remains independent of the analysis target.

EMERALD also introduces a target-independent code generation utility that allows one to automatically produce the library interfaces necessary to integrate a P-BEST expert system into the EMERALD monitor infrastructure. This utility effectively relieves the creator of a resource object from dealing with the internal operation of the eXpert code base, even when redirecting the eXpert to a completely new event stream. This automated generation utility both enhances the rapid integration of eXpert to new analysis targets, and simplifies the process of augmenting the rule base with new heuristics. The basic operation of an eXpert analysis engine is as follows:

1. On startup, eXpert is initialized and its interface routine waits for messages on one or several transports, as specified in the configuration files of the resource object.

2. When an event record is received in the form of an EMERALD message, the message is matched against an interface data structure associated with the ptype definition in the eXpert's P-BEST factbase.

3. The message content is transferred to the interface data structure, which in turn is used to assert a fact into the expert system factbase.

4. The eXpert interface component hands over control to the expert system inference engine.

5. If a rule is fired, in which the consequent specifies that an alert shall be generated, the alert is propagated back to the analysis engine's interface component, which in turn composes and sends the alert on to the EMERALD resolver. The resolver operates as the monitor's decision engine, and can invoke local responses based on the alert or propagate the alert on to subscribers of the monitor's results (including administrative display interfaces).

6. When there are no more rules that can fire, the expert system returns control to the interface routine that again starts waiting for incoming messages.

In the following section, we discuss examples of how eXpert can be used to analyze very different types of event streams.

2.4 eXpert rule development examples

Throughout its usage, P-BEST inference engines have implemented a variety of intrusion detection rule sets for detecting and responding to numerous forms of malicious activity. We describe the application of P-BEST in reasoning about attacks represented in two data streams: Solaris 2.5.+ audit trails, and TCP/IP packet streams. The examples illustrate the declarative style of the language, and how event streams can be represented and analyzed.

2.4.1 Examples of BSM audit trail analysis

The first example of an event stream to be analyzed is the audit trail produced by the Solaris Basic Security Module (BSM) from Sun Microsystems [55]. The audit records are normally saved in a file, but we have developed a BSM collection unit that receives audit records from the OS kernel in real time, and formats and sends each record as an EMERALD message to the target monitor for analysis.

For all the rules that analyze BSM data, there is a ptype called `bsm_event` into which the relevant fields from incoming messages are mapped. There is also a rule that has highest priority and copies the time of every incoming `bsm_event` fact into a new `time` fact, and finally a rule with lowest priority that removes the `bsm_event` fact after all the other rules have had a chance to look at it. For the sake of brevity, these ptype definitions and administrative rules are omitted from the examples.

Failed authentication attempts

As an example of the declarative programming paradigm that P-BEST supports, we present a set of rules that are designed to detect a number of failed authentication attempts within a certain time window. The example illustrates how facts are created in rule consequents to keep state information between incoming events, and how the rule designer can make sure that facts are removed from the factbase when they are no longer needed.

Let us assume that we want to raise an alert if `x` user authentication failures occur within `y` seconds for a monitored target. A user authentication failure is defined as the case when either an invalid username or an invalid password is given to one of the programs *login, telnet, rlogin, rshd,* or *su.* To accomplish this, we may employ the rule set presented in Table 2.1, which is described as follows:

• `A1, A2`: For every incoming event that is a user authentication failure, save the event information in a `bad_login` fact and increment the counter for current bad logins (`current_bl_cntr`) by 1. The reason for having two rules is to separate the case where the username is invalid (`A1`) from the case where the username is valid but the password is invalid (A2). In the latter case, we want to include the username in the information we save and therefore need a rule consequent that is different from the former case where there is no username reported in the audit record.

• A3: When the `current_bl_cntr` counter has the value x, send an alert and create a `max_bl_reached` fact to indicate that the authentication failure threshold was reached.

• A4: If there exists a `max_bl_reached` fact, then loop through all saved `bad_login` facts. For every `bad_login` fact, print the information contained in the fact to a log file and delete the fact from the factbase.

• A5: If there exists a `max_bl_reached` fact but no `bad_login` facts (i.e., they were all printed and deleted by rule A4), then delete the `max_bl_reached` fact from the factbase.

• A6: If there exists a `bad_login` fact, but no `max_bl_reached` fact, and the difference between the `bad_login` timestamp and the current event timestamp is more than y seconds, then delete the `bad_login` fact from the factbase and decrement the `current_bl_cntr` by 1.

Buffer overrun attacks

Buffer overrun attacks are a common way for attackers to gain super-user privileges after first breaking into an unprivileged user account. Typically, a privileged (*setuid* to *root*) program is called with an extremely long and carefully crafted argument that overflows memory buffers and alters the program execution [8]. In principle, it would require a fair amount of programming skills and patience to exploit a buffer overrun vulnerability, but ready-to-use exploit programs that can be downloaded from Internet sites give immediate super-user access when executed. Here, we present an example of a simple heuristic P-BEST rule that detects the behavior of most of the exploit programs. For example, it has been tested against buffer overrun exploits that are based on subverting Solaris 2.5 *eject, fdformat, ffbconfig,* and *ufsrestore.*[1]

The heuristic rule is based on the following observations of the audit trail characteristics of common buffer overrun exploits:

- We can detect the attack by analyzing a single *exec* system call audit record, as suggested in [6].
- To determine that the *exec* call concerns a *setuid* program (otherwise, it would not be a target for attack), we simply match only the audit records for which the *effective user id* and *real user id* fields are different.
- The argument passed to the *exec* call is relatively long (because it must overflow a buffer and contain executable code), making the length of the entire audit record significantly exceed the length of almost all normal *setuid exec* calls.

[1]Numerous additional buffer overrun attacks employ an attack strategy identical to that of the four attacks discussed here. All should be subject to detection by this rule.

Table 2.1: Rule set for detection of failed authentication attempts.

```
rule[A1(*):
   [+e:bsm_event^A12]
   [?|e.header_event_type   == 'AUE_login  ||
      e.header_event_type   == 'AUE_telnet ||
      e.header_event_type   == 'AUE_rlogin ||
      e.header_event_type   == 'AUE_rshd   ||
      e.header_event_type   == 'AUE_su]
   [?|e.return_return_value == 'INVALID_USER]
   [+cc: current_bl_cntr]
   [-max_bl_reached]
==>
   [+bad_login |
      timestamp    = e.header_time,
      audit_seq_no = e.msequenceNumber,
      username     = "invalid username",
      command      = e.header_command,
      etype        = e.header_event_type,
      hostname     = e.subject_hostname,
      portID       = e.subject_port_id,
      processID    = e.subject_pid,
      textList     = e.textList]
   [/cc| value += 1]
   [$|e:A12]
]
```

```
rule[A2(*):
   [+e:bsm_event^A12]
   [?|e.header_event_type   == 'AUE_login  ||
      e.header_event_type   == 'AUE_telnet ||
      e.header_event_type   == 'AUE_rlogin ||
      e.header_event_type   == 'AUE_rshd   ||
      e.header_event_type   == 'AUE_su]
   [?|e.return_return_value == 'INVALID_PWD]
   [+cc: current_bl_cntr]
   [-max_bl_reached]
==>
   [+bad_login |
      timestamp    = e.header_time,
      audit_seq_no = e.msequenceNumber,
      username     = e.subject_runame,
      command      = e.header_command,
      etype        = e.header_event_type,
      hostname     = e.subject_hostname,
      portID       = e.subject_port_id,
      processID    = e.subject_pid,
      textList     = e.textList]
   [/cc| value += 1]
   [$|e:A12]
]
```

continues on next page

continued from previous page

```
rule[A3(*):
    [-max_bl_reached]
    [+cc:current_bl_cntr | value == 'x]
    [+ts:time^A3]
==>
    [!|printf("ALERT: Max Bad Logins \n")]
    [+max_bl_reached | value = 1]
    [$|ts:A3]
    [!|EXpertReport('eXpertMessagePointerString,
       1042, "description", 'pTypeString,
       "MAX LOGIN ALERT",
       "ruleName", 'pTypeString, "A3", "")]
]
```

```
rule[A4(*):
    [+max_bl_reached]
    [+bc:bad_login]
    [+cc:current_bl_cntr]
==>
    [!|printf("(%s): %s from %s on %s port %d, \
      PID = %d, time = %d, seq no = %d \n",
      bc.textlist, bc.command, bc.username,
      bc.hostname, bc.portID, bc.processID,
      bc.timestamp, bc.audit_seq_no)]
    [/cc|value -= 1]
    [-|bc]
]
```

```
rule[A5(*):
    [+mx:max_bl_reached]
    [-bad_login]
==>
    [-|mx]
]
```

```
rule[A6(*):
    [+ts:time^A6]
    [-max_bl_reached]
    [+bc:bad_login]
    [+cc:current_bl_cntr]
    [?|(ts.sec - bc.timestamp) > 'y]
==>
    [/cc|value -=1 ]
    [-|bc]
    [$|ts:A6]
]
```

- By necessity of the applicable hardware (Sun and Intel), the *exec* argument contains binary opcodes in the range of ascii control characters. While such a property may not necessarily hold on all possible hardware platforms, this heuristic works exceptionally well for our purposes.

The P-BEST rule that uses the observations above to detect buffer overrun attacks is shown in Figure 2.6. This simple heuristic rule is not a foolproof way to detect all possible buffer overrun attacks, but it is remarkably efficient in terms of coverage and correctness; it detects most common attacks and has not produced any false positives when tested on a collection of more than 35 million audit records in which the location of buffer overflow attacks was known *a priori*.

```
rule[BSM_LONG_SUID_EXEC(*):
   [+e:bsm_event]
   [?|e.header_event_type == 'AUE_EXEC ||
      e.header_event_type == 'AUE_EXECVE]
   [?|e.subject_euid != e.subject_ruid ]
   [?|contains (e.exec_args, "^\\") == 1]
   [?|e.header_size > 'NORMAL_LENGTH]
==>
   [!|printf("ALERT: Buffer overrun attack \
      on command %s\n", e.header_command)]
]
```

Figure 2.6: A heuristic rule for detecting common buffer overrun attacks.

To determine a suitable value for the NORMAL_LENGTH threshold parameter, we have analyzed in the order of 4 million audit records representing normal system usage (of which more than 29 thousand were *exec* events) in addition to audit records representing common buffer overrun attacks. This analysis gave the following results:

- All the attacks we tested produce an *exec* audit record with a record length of at least 500 bytes.
- Only 0.15 percent of the normal *exec* audit records were longer than 400 bytes.

Consequently, by setting the threshold to 400 and adding the conditions for *setuid* and control characters, false positives are effectively eliminated while exploits of the described type are detected.

2.4.2 Network-based traffic analysis

In addition to its extensive application to the area of audit trail analysis, P-BEST is now being applied to the analysis of network traffic streams. This work includes the analysis of TCP/IP packet streams for low-level TCP- and IP-layer attacks (i.e., attacks that target vulnerabilities at the transport layer and below) as well as higher-layer attacks involving vulnerabilities of application-layer (or network service-layer) protocols, such as FTP, SMTP, and HTTP.

Attack description: SYN flood attack

The SYN flood attack is a denial-of-service attack that prevents the target machine from accepting new connections to a given IP port [52]. Briefly, the attack exploits a resource exhaustion vulnerability in the way operating systems handle TCP/IP connections. A TCP/IP connection is established through a three-step handshake, in which the client sends a SYN packet, followed by the server responding with a SYN-ACK packet, which is then acknowledged by the client with an ACK packet. Of course, by no means is there an expectation that all TCP/IP handshakes run to completion. When the SYN packet is received, the server allocates an entry in a finite queue of pending connections. We refer to this stage as a *half-open* connection. The queue entry will either be released when the final ACK is received by the server, or the server will proceed to timeout the incomplete handshake and release the entry.

An attacker can exploit the TCP/IP connection logic by initiating a series of SYN packet connection requests to a server, but not completing the handshakes with an ACK packet. Internally, the server's queue of pending connections for the port will eventually be exhausted and will not be released until the timeout periods for the unfinished connections expire. As a result, subsequent connection requests to the server that occur while the connection queue is full will be dropped, effectively denying access to the server by other legitimate clients.

Event stream format

The requirements for detecting the occurrence of a SYN flooding attack against a host are rather minimal. From the perspective of TCP/IP traffic monitoring, the analysis engine need only monitor SYN-ACK and ACK packet exchanges to identify incomplete TCP/IP handshakes. In this example, the traffic monitor is placed on a segment of the network capable of observing traffic to and from the analysis target (the host being monitored). All SYN-ACK packets sent from—and ACK packets sent to—the analysis target are recorded, and the following event record is derived:

```
Connection Event Format:
 <Event_Type> <Timestamp> <Seq_ID> <Client_ID>
```

The Event_Type field is simply a binary flag, which indicates whether the packet has its SYN and ACK flags enabled (which we can denote with 0), or only the ACK flag enabled (denoted by 1). The timestamp is a numeric encoding of the time at which the packet is observed from the monitor. The sequence ID represents the TCP Sequence ID field, which is used to associate client requests with server replies. Last, the Client_ID can be used to identify the client that initiated the connection. The Client_ID is not critical for detection, and in all likelihood will not be reliable (i.e., attackers will manufacture IP packets with bogus IP source addresses). Nevertheless, we may choose to capture such information as the IP address and port number of the client packet for reporting purposes only.

Table 2.2: Facts for TCP SYN flood detection.

1	`ptype[conn_event`	`ptype[open_conn`	`ptype[bad_conn`
2	`e_type:integer,`	`expired:integer,`	`count:integer]`
3	`sec:integer,`	`sec:integer,`	
4	`seq_id:integer,`	`seq_id:integer,`	
5	`client_ID:string]`	`client_ID:string]`	

P-BEST fact type definitions

Table 2.2 illustrates the ptype definitions of three example facts that are specified for use in performing the TCP SYN flooding analysis. The first ptype, `conn_event`, is used to assert the connection event described in the connection event record format discussed above. As connection events are captured by the network monitor, their fields can be mapped (one to one) to the fields of the `conn_event` ptype, and the `conn_event` ptype is then asserted into the factbase of the SYN flood eXpert. The `open_conn` ptype is used to construct facts regarding half-open connections that are pending completion of the TCP/IP handshake. Note that although we use the shorthand name `open_conn`, the fact actually represents the assertion that a TCP *half-opened* connection has been observed. The fields of the `open_conn` contain the TCP sequence ID of the pending connection, a `client_ID` string (as discussed above), the timestamp as copied from the connection event, and an expired flag used for garbage collection by the production rules. Last, the bad connection fact, `bad_conn`, maintains a running count of the number of bad connection requests detected through the observations of SYN-ACK and ACK packages between the analysis target and external clients.

Example P-BEST rules for SYN flood detection

The following illustrates one inference strategy that P-BEST can employ for deducing a TCP SYN flooding attack, using the fact definitions defined above. In addition, a few constants are referenced from the rule set, and are defined as follows:

- `max_bad_conns`: Number of bad connections tolerated before SYN flood alert.
- `expire_time`: Amount of time to wait on ACK before a connection is declared a bad connection.
- `bad_conn_life`: Number of seconds that a bad connection fact will live before being released.

Abstractly, the rules attempt to identify half-open TCP connections that expire beyond a user-defined waiting period. As we assert half-open connection facts into our factbase, we must include logic to recognize both when the connections are successfully completed and when half-open connection expire beyond the user-defined waiting period, from which we deduce the occurrence of a bad connection. SYN flood attacks will result in excessive bursts of bad connections, which we monitor with rules that maintain a running count of bad connections over a sliding window of time. When the number of bad connections exceeds our maximum tolerance for bad connections within our sliding time window, we raise an alert to denote the burst of noncompleted connection requests. The following is a brief summary of the rule set shown in Table 2.3.

- `create_open_conn`: determines whether the event connection represents a SYN-ACK packet (from the monitor target). If so, the rule asserts a new fact into the factbase called **open_conn**, which records the TCP sequence number, the timestamp at which this half-opened connection was first observed, an expired flag to indicate when the half-open connection exceeds a time threshold, and the `client_ID`.

- `destroy_open_conn`: removes an open connection fact when the corresponding ACK packet is received from the client.

- `ignore_spurious_acks`: removes events involving ACK packets that are not associated with a specific SYN-ACK pending connection. In practice, such packets are normal.

- `first_bad_conn`: This and the following rule manage a running count of the set of bad connections observed by the inference engine. They are driven by time facts (line 24) that are used to monitor whether there exists a half-open connection that has exceeded the `expire_time` limit. This rule is applied once, to the first **open_conn** fact encountered that is older than `expire_time`. Its consequent creates the **bad_conn** fact, which initializes the bad connection counter upon the first encountered expired connection. Note that the antecedent line 25 evaluates to false once the **bad_conn** fact has been initialized. In addition, the rule marks the **open_conn** fact as expired (line 30), which is consulted by `free_bad_open_cons` when performing garbage collection.

- `add_to_bad_cons`: is applied while the total number of **bad_conn** facts is less than the maximum tolerated. If an **open_conn** fact timestamp exceeds the expiration time and the fact has not been counted earlier, then the **bad_conn** count is incremented, and the expired flag for the **open_conn** fact is set.

- `max_open_cons`: is applied when the maximum number of **bad_conn** facts is encountered during a burst of `bad_conn_life` time units. If a **bad_conn** count reaches the maximum tolerated **bad_conn** facts, the consequent initiates a SYN flood alert, and resets the bad connection count.

- `free_bad_open_cons`: limits the amount of time that a bad open connection is counted against the system. The `bad_conn_life` variable provides a user-defined length of time with which a bad connection is considered relevant to the bad connection count. This variable effectively represents the burst duration for accumulating bad connections. Once an open connection exceeds the `bad_conn_life`, then it is removed and the bad connection count is reduced.

- `del_alerted_cons`: deletes the half-open connections that have caused an alert.

Table 2.3: Rule set for detection of TCP SYN flood attacks.

```
rule[create_open_conn(*):
   [+ev:conn_event|e_type == 0]
==>
   [+open_conn |seq_id = ev.seq_id,
               sec = ev.sec,
               expired = 0,
               client_ID = ev.client_ID]
   [-|ev]
]
```

```
rule[destroy_open_conn(*):
   [+ev:conn_event|e_type == 1]
   [+oc:open_conn|seq_id == (ev.seq_id - 1),
                 expired == 0]
==>
   [-|oc] [-|ev]
]
```

```
rule[ignore_spurious_acks(*):
   [+ev:conn_event|e_type == 1]
   [-open_conn|seq_id == (ev.seq_id - 1)]
==>
   [-|ev]
]
```

```
 rule[first_bad_conn(*):
   [+ts:time]
   [-bad_conn]
   [+oc:open_conn|expired == 0]
   [?|(ts.sec - oc.sec) > 'expire_time]
==>
   [+bad_conn|count = 1]
   [/oc| expired = 1]
]
```

continues on next page

continued from previous page

```
rule[add_to_bad_cons(*):
   [+ts:time]
   [+oc:open_conn|expired == 0]
   [?|(ts.sec - oc.sec) > 'expire_time]
   [+bc:bad_conn|count < 'max_bad_conns]
==>
   [/bc|count += 1]
   [/oc|expired = 1]
]
```

```
rule[max_open_cons(*):
   [+ts:time]
   [+oc:open_conn|expired == 0]
   [?|(ts.sec - oc.sec) > 'expire_time]
   [+bc:bad_conn|count == 'max_bad_conns]
==>
   [!|syn_alert("SYN Attack: Last Host %s.\
      SeqID = %d. Time = %d",
       oc.client_ID, oc.seq_id, oc.sec)]
   [/bc|count = 0]
   [/oc|expired = 1]
]

```

```
rule[free_bad_open_cons(*):
   [+ts:time]
   [+bc:bad_conn]
   [+oc:open_conn|expired == 1]
   [?|(ts.sec - oc.sec) > 'bad_conn_life]
==>
   [-|oc]
   [/bc|count -= 1]
]
```

```
rule[del_alerted_cons
   [+oc:open_conn|expired == 1]
   [+bad_conn|count == 0]
==>
   [-|oc]
]
```

2.5 Performance

A variety of factors influence the amount of time required to process records through a P-BEST-based signature analysis engine. In this section, we briefly discuss some of these factors and summarize several performance measurements in analyzing both Solaris audit records and TCP packets through an EMERALD eXpert P-BEST engine. These measurements are intended to reflect the pure processing time required by the eXpert in receiving events, translating and asserting the events into the eXpert factbase, processing the events through the inference engine, and handling alert reporting.

The measurements exclude the processing time added to the system for event generation; that is, they exclude the impact to system resources in audit record generation or the capturing and filtering of TCP packets. It is difficult to estimate the daily expected volumes of audit and network traffic across a computing environment, in that such statistics are directly dependent on the structure of the computing environment, network topology, and behavior and size of the user community. Furthermore, the EMERALD architectural model lends itself well to the separation of the event generation and collection components from the analytical engines, which could in fact operate in parallel on separate hosts.

The performance measurements were collected on a FreeBSD 2.2.6 host computer system using a Pentium II 333 MHz processor with 128 MB RAM. In addition to the processing capabilities of the host platform, several factors significantly influence the overall performance of the analysis engine. For example, the average record size and total event stream size dictate the amount of I/O overhead required. As each event is asserted by the rule base, the antecedent evaluation also impacts performance: the sheer number of rules to evaluate, as well as the complexity of each antecedent evaluation, significantly influence event processing throughput. Consequent activation is also a consideration, as is the management of derived facts that are asserted during the analysis.

Table 2.4 presents a summary of three analyses performed on 1- and 5-day collections of Solaris 2.5.1 audit records and TCP packet streams. The audit and TCP data sets were collected by MIT Lincoln Laboratories, and made available for the DARPA Intrusion Detection Evaluation Program. The BSM audit logs analyzed here represent the simulated usage of a server with 43 users over one 24-hour period and 44 users over a 5-day workweek, with minimal filtering. While it is difficult to generalize what such loads imply for other computing environments, the data set is representative of the volume and type of audit activity observed during a prolonged study of several Air Force local area networks.

The first row in Table 2.4 summarizes the performance of an EMERALD eXpert implementing the buffer overflow rule presented in Section 2.4.1, which is roughly able to apply this rule to 24 hours of audit data (over 1 million audit records) in 4 minutes, and 120 hours of audit data (4.2 million audit records) in under 16 minutes. In the second row, we present an eXpert with a more extensive collection of 28 rules. These rules implement 16 sets of Solaris BSM intrusion detection heuristics, including threshold analyses, immediate at-

Table 2.4: Performance of sample BSM and TCP analysis engines.

	24 hrs BSM 43 users 365 MB total 1.1 million recs	120 hrs BSM 44 users 1.41 GB total 4.2 million recs	24 hrs IP 496 connects 331 MB total 83,002 recs	120 hrs IP 1,343 connects 1.3 GB total 352,445 recs
1 rule set *2 rules* *buffer overrun*	4:10 min:sec	15:41 min:sec	—	—
16 rule sets *28 rules* *various intrusions*	8:09 min:sec	30:53 min:sec	—	—
1 rule set *12 rules* *TCP SYN flood*	—	—	1:33 min:sec	3:02 min:sec

tack recognition, process subversion detection, and illegal file access recognition. While the knowledge base of this second eXpert represents an increase of fourteenfold over the 2-rule eXpert system in the first row, it introduces only a twofold increase in the overall processing time of the 1- and 5-day data sets. In this computing environment, the 16 rule sets can process the full 5-day data set in just over 30 minutes; this represents a small fraction of the overall audit generation time.

The third and fourth columns of Table 2.4 present an analysis of TCP/IP traffic through a gateway that provides service between an internal domain of 4 servers and 20 workstations, and an external untrusted network. The third row of Table 2.4 summarizes the performance of the TCP SYN flood detection rules presented in Section 2.4.2 (with a few additional administrative rules). Here, a server was selected for analysis, and all TCP packets sent to and from it were monitored for 24 and 120 hours, during which 496 and 1,343 connections were observed over 24 and 120 hours, respectively. The SYN Flood eXpert monitored only those TCP packets targeted for the host of interest in which the SYN or ACK flags were enabled. The filtering out of unnecessary packets is critical to managing the performance of a real-time signature analysis engine, and in the SYN flooding case, the criteria for analysis excludes all packets that are not directly involved in the TCP handshake. In our simulated analysis, the SYN Flood eXpert is capable of performing the 24-hour packet analysis in 1.5 minutes, and the 120-hour analysis in 3 minutes.

2.6 Related work

P-BEST has evolved over a substantial lineage of intrusion detection projects, which include MIDAS, IDES, NIDES, and now the EMERALD eXpert. It represents a very early example of the application of a forward-chaining rule-based expert system to the problem of misuse detection in computer system activity logs. However, P-BEST is by no means the only system to have applied rule-based expert system techniques to detecting misuse in computing environments.

Several other systems have been developed that also center around the use of forward-chaining inference logic, and have applied a variety of techniques for representing the underlying heuristics used to represent misuse. The ASAX (Advanced Security and Audit Trail Analysis on UniX) project [21], produced a highly specialized rule-based programming language called RUSSEL (Rule Based Sequence Evaluation Language), which provides a combination of procedural and rule-based programming constructs to reason about activity in Unix audit trails.

The University of California at Santa Barbara proposed the use of state transition diagrams to model the sequence of operations and state changes that occur during the execution of a penetration [45]. This technique was prototyped for SunOS 4.1.3+ and Solaris audit trails in a tool called the Unix State Transition Analysis Tool (USTAT) [24]. While it did not represent its knowledge base using production rules, USTAT was architected as a classic expert system, with an inference engine, knowledge base, factbase, and separate decision engine. Another system, called IDIOT (Intrusion Detection In Our Time), took a similar graphical approach to the analysis of signature operations, but used Colored Petri-nets to model its analysis of the patterns of execution represented in an event stream [30].

Wisdom and Sense (W&S) [58] and NADIR [26], both from Los Alamos National Laboratory, are further examples of intrusion detection systems that employed rule-based analyses to identify known malicious activity. In the case of W&S, the anomaly detection component was also implemented as a rule base. The signature analysis component was combined into the same rule base to represent site-specific policies, expert penetration rules, and other administrative data. NADIR's expert rule base consists of penetration rules that are developed by interviewing and working with security personnel.

Last, it is important to recognize a continuing growth in the number of commercial products that provide forms of signature analysis for various computing environments. Given the proprietary nature of these systems, it is difficult to understand which have chosen hard-coded narrow solutions to their problem sets, and which have chosen more broad techniques that may be portable beyond their current customers' needs.

2.7 Limitations

In Section 2.2 we attempted to summarize how and why forward reasoning systems provide a good foundation for modeling known abusive activity represented in an event stream. There are, of course, limitations that are fair to point out with respect to this general method. In our own system, antecedent evaluation is absolute, and less capable in environments where uncertainty, incompleteness, or inaccuracies exist within the event stream content. Other reasoning systems can provide some options for handling belief and uncertainty within the analysis framework [20]. In the presence of incomplete data, backward reasoning systems can operate in a diagnosis mode to seek out collaborative evidence of problems, and furthermore provide quantitative probabilities based on "evidence to date" that a certain problem is the culprit responsible for the presence of given symptoms. Such reasoning capabilities could be valuable if applied well to the intrusion detection domain.

In addition to event stream inadequacies, heuristics presuppose the existence of detailed insight into that which constitutes abusive system activity. The problem of recognizing and responding to *unknown* malicious phenomena is extremely difficult, and not directly addressed under signature analysis. Only in the cases where it is possible to look for certain *results*—rather than explicit action sequences leading to those results—does signature analysis have a chance to detect new attack methods. For example, if an anonymous user causes the deletion of a file from our FTP server, we can detect this result without knowing exactly how the attack was carried out.

Other techniques that attempt to understand *normal* system operation and to provide quick recognition of anomalous activity have been proposed; statistical profiling [16], neural networks [15], and sequence analysis [19]. The intent of these systems is to maximize the points at which anomalous activity corresponds to malicious activity, which as a general property does not always hold. In addition, attempting to maximize such systems' sensitivity to malicious activity also tends to increase their sensitivity to inane anomalies.

2.8 Conclusions

We have presented the operation of a production-based expert-system toolset, and its application to the problem of computer and network signature-based intrusion detection. P-BEST has had considerable exposure to the intrusion detection problem domain over the past decade, under the MIDAS, IDES, and NIDES projects, and now within the EMERALD eXpert. P-BEST has been employed on a Symbolics processor for handling Multics audit records, SunOS 4.1.+, Solaris 2.5.+, FreeBSD, and Linux for real-time audit trail analysis, accounting log analysis, and TCP/IP packet analysis.

We presented details of the P-BEST production rule specification language, and illustrated its use with example rule sets for detecting misuse in Solaris 2.5.+ audit trails and TCP/IP packet streams. We also discussed the performance of

P-BEST inference engines in analyzing millions of events, which illustrates that P-BEST has been—and continues to be—useful in live monitoring of computer and network operations.

P-BEST is currently being used for laboratory exercises in one university course on applied computer security, where students are guided through its usage and assigned rule development tasks for analyzing given intrusions. We have demonstrated that the P-BEST language is not too complex for beginners to employ, and is efficient for supporting the iterative development of increasingly complex inference logic for automated reasoning about misuse in computer and network operations.

Chapter 3

Network-based Detection

This chapter describes the development of EMERALD intrusion detection monitors that analyze intercepted network traffic.

3.1 Introduction

In the commercial arena, Intrusion Detection Systems (IDSs) that take their input data from network traffic are so dominating that many people equate intrusion detection with network traffic analysis. Although we argue that data for intrusion detection should be collected simultaneously from several different abstraction layers and sources, network traffic is indeed an important data source for intrusion detection and, consequently, we have expended significant efforts on developing EMERALD monitors for network data.

In contrast to what is commercially available today, EMERALD network monitors are mainly focused on detecting events that manifest themselves in multiple packets, and in transactions and user sessions. The EMERALD network sensors also perform varying degrees of alert aggregation to prevent flooding of the operator console with, for example, alerts about scans and probes that can occur with high frequency at many Internet sites.

The *eXpert-Net* knowledge base represents a comprehensive collection of intrusion detection heuristics for network traffic analysis. Where possible, rules are implemented to provide the most general coverage for misuse detection and security policy violations to cover the widest range of attack classes possible from network-based analysis. These rules have been extensively tested for their ability to recognize the intrusive activity described below, and also to avoid false positives.

For every detection rule, the user-configurable parameters that control the behavior of the rule are listed and described, with the following notation:

- RULE_NAME: Description
  - ▶ PARAMETER_NAME: Description

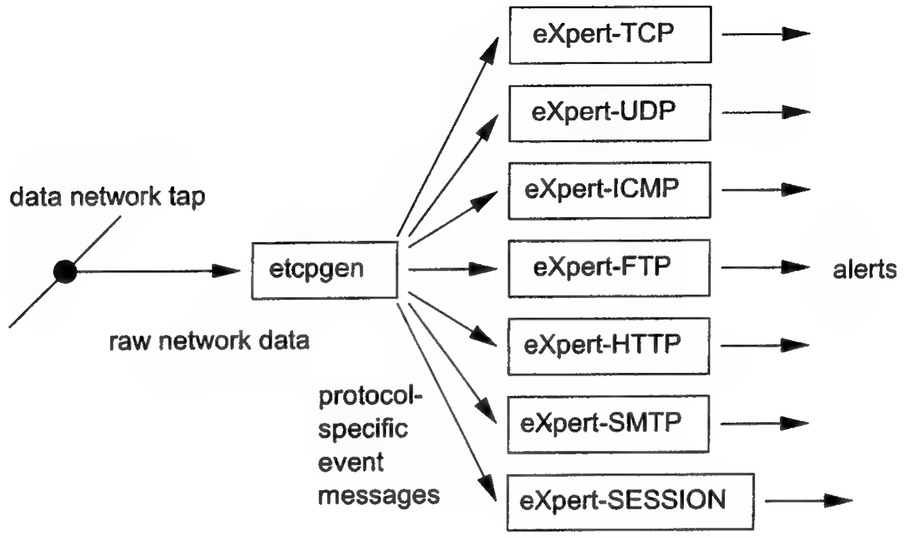


Figure 3.1: Data flow architecture for eXpert-Net

3.2 Architecture

The data flow architecture of the *eXpert-Net* package is somewhat similar to that of *eXpert-BSM* (described in Chapter 4) in that it has a preprocessing component that interprets the raw event data and produces EMERALD messages for the analysis engines, but it is different in that it contains multiple expert-system-based analysis engines. The dataflow architecture is outlined in Figure 3.1.

The preprocessing *etcpgen* component plays an important role in our network traffic analysis. It uses *libpcap* [38] for low-level network event capture, and then interprets a number of protocols and performs reconstruction of transactions and sessions. For example, for HTTP it combines each client request and corresponding server reply into a single message that describes the HTTP transaction. For protocols such as FTP, telnet and other that have longer user-oriented sessions, *etcpgen* keeps track of session-related data and makes sure that each message contains the relevant session data fields.

3.3 Transport protocols

The *eXpert-Net* package performs analysis of the transport-level protocols TCP, UDP, and ICMP. What is common for this analysis is that each network packet (after IP-level fragmentation reassembly) will result in a message sent to the corresponding analysis engine. However, this does not restrict the analysis to single-packet inference, as each analysis engine has sets of rules that perform stateful multimessage analysis.

3.3.1 eXpert-TCP rulesets

- TCP_LAND: The Land attack uses a TCP SYN packet whose source IP address is equal to its destination IP address and its source port number is equal to its destination port number. This attack can cause denial of service.

- TCP_TEARDROP: This rule detects overlapping TCP fragments, which are a characteristic of the Teardrop attack. The Teardrop attack can cause denial of service.

- TCP_SUSPICIOUS_PACKET: This rule looks for TCP packets that have both SYN and FIN flags set and have a source port number equal 0 or 65535. It is likely that these packets belong to a probe or an ill-intentioned program.

- TCP_CHARGEN_TO_ECHO: This rule detects a denial-of-service attack that involves the CHARGEN and the ECHO services. In this attack, packets are sent between the CHARGEN port of a machine to the ECHO port of another machine. Because these services always send back a reply when they receive a request, an infinite loop of network traffic is formed.

  ► TCP_PORT_LOOP_TIMEOUT defines the amount of time (in seconds) this rule starts monitoring two machines for a loop of network traffic again after the previous one has ended.

- TCP_ECHO_TO_ECHO: Similar to TCP_CHARGEN_TO_ECHO, except that this rule looks for TCP traffic between the ECHO port of a machine and the ECHO port of another machine.

  ► TCP_PORT_LOOP_TIMEOUT defines the amount of time (in seconds) this rule starts monitoring two machines for a loop of network traffic again after the previous one has ended.

- TCP_FINGER_REDIR: This rule fires when a request whose argument contains "@@@" is sent to the FINGER port of an internal machine. This attack can cause denial of service.

- TCP_FINGER_RESERVED_NAME: This rule detects probes to reserved accounts (e.g., root) using the FINGER service.

- TCP_FINGERD_OVERFLOW: This rule fires when a long FINGER request containing many control characters is observed. These requests may indicate a buffer overflow attack.

  ► TCP_FINGERD_OVERFLOW_LEN is the maximum allowable length of a client-supplied argument without triggering this rule.

  ► TCP_DATA_MAX_CONTROL_CNT specifies the maximum allowable number of control characters in a client-supplied argument without triggering this rule.

- TCP_NAMED: This rule detects buffer overflow attacks against NAMED servers. It compares a list of suspicious patterns against DNS queries that are sent to the DNS port of an internal machine.

  ▶ TCP_NAMED_OVERFLOW_PATTERN contains a list of patterns that appear in DNS queries of NAMED buffer overflow attacks.

- TCP_IMAPD_BO: This rule fires when a long IMAP request containing many control characters and containing "LOGIN" or "AUTHENTICATE" is observed. This request may indicate a buffer overflow attack on an IMAP server.

  ▶ TCP_IMAPD_OVERFLOW_LEN defines the maximum allowable length of an IMAP request without triggering this rule.

  ▶ TCP_DATA_MAX_CONTROL_CNT specifies the maximum allowable number of control characters in an IMAP request without triggering this rule.

- TCP_PORT_SWEEP: When the number of distinct ports scanned within a certain time window exceeds a certain threshold, this rule will fire.

  ▶ TCP_PORT_SWEEP_THRESHOLD specifies the maximum number of ports scanned before this rule is triggered.

- TCP_FIN_ACK_SCAN: This rule fires when the number of distinct ports scanned using TCP FIN packets exceeds a certain threshold. FIN scans are considered as stealthy.

  ▶ TCP_PORT_SWEEP_THRESHOLD specifies the maximum number of ports scanned before this rule is triggered.

- TCP_FIN_ADDR_SCAN: This rule fires when the number of distinct IP addressed scanned using TCP FIN packets exceeds a certain threshold.

  ▶ TCP_ADDR_SWEEP_THRESHOLD defines the maximum number of hosts scanned before this rule is triggered.

- TCP_ADDR_SWEEP: This rule fires when the number of distinct IP addresses scanned from a single source within a certain time window exceeds a certain threshold.

  ▶ TCP_ADDR_SWEEP_THRESHOLD defines the maximum number of hosts scanned before this rule is triggered.

- TCP_TELNET_FLOOD: This rule fires if the number of requests to the TELNET port within a certain time window exceeds a certain threshold.

  ▶ TCP_TELNET_FLOOD_THRESHOLD defines the maximum number of times to which the TELNET port is connected before this rule is triggered.

- TCP_DNS_ZONE_TRANSFER: This rule fires when a zone transfer from an external host to an internal DNS server is observed. A DNS zone transfer may indicate an intelligence-gathering operation to gain information about the hosts in an organization.

  ► TCP_MAX_DNS_CHAT_LIFE defines the size of the time window (in seconds) during which DNS transactions are monitored by this rule.

  ► TCP_MAX_DNS_ACK_CNT defines the number of TCP ACK packets observed for a DNS session before this rule is triggered.

- TCP_BRKILL: This rule detects the following sequence of TCP events: (1) a TCP packet that has the PUSH and ACK flags set and its sequence number equals zero; (2) many TCP packets with the RESET flag set sending to the same host as that in (1). This attack can cause denial of service.

  ► TCP_MAX_BRKILL_FACT_LIFE defines the size of the time window (in seconds) during which a TCP connection is monitored by this rule.

  ► TCP_MAX_BRKILL_RESETS defines the maximum allowable number of TCP RESET packets before this rule will fire.

- TCP_PORT_FLOOD: This rules fires when the number of packets sent to a port within a certain time window exceeds a certain threshold. Certain ports that usually have a lot of traffic can be excluded from this analysis.

  ► TCP_SINGLE_PORT_FLOOD_THRESHOLD is the maximum number of times to which a single port is connected before this rule will fire.

  ► TCP_HIGH_TRAFFIC_PORTS defines a list of ports that normally are exposed to a large number of connections, and therefore should be ignored by this rule.

- TCP_POP_DICT: This rule detects password guessing attempts against a POP server. This rule fires when the number of failed login attempts from a host to a POP server exceeds a certain threshold.

  ► TCP_POP_DICT_THRESHOLD defines the maximum allowable number of failed POP login attempts before this rule will fire.

- TCP_MSCAN: This rule fires when an attacker uses a scanning tool called MSCAN to probe a host.

  ► TCP_MSCAN_RESET_FACT_LIFE defines the amount of time (in seconds) a detected MSCAN attack from a host to another is "forgotten"; after which the expert starts looking for another MSCAN instance involving the same source IP address and the same destination IP address.

  ► TCP_MAX_MSCAN_FACT_LIFE defines the size of the time window (in seconds) during which a potential MSCAN attack is monitored.

- TCP_NTINFOSCAN: The NTInfoScan is a NetBIOS-based port scanning tool against Windows NT machines.

► TCP_MAX_NTINFOSCAN_FACT_LIFE defines the size of the time window (in seconds) during which a potential NT InfoScan is monitored.

► TCP_FTP_INFO_SCAN_PATTERN contains a list of patterns that appear in the FTP requests used by InfoScan.

► TCP_HTTP_INFO_SCAN_PATTERN contains a list of patterns that appear in the HTTP requests used by InfoScan.

- TCP_WINNUKE: This rule fires when a TCP packet whose destination port equals a certain number (e.g., 139) and has its URGENT POINTER flag set is sent to a Windows machine. This attack can potentially halt a Windows machine.

- TCP_REMOTE_RLOGIN: This rule detects remote login attempts from an external machine.

- TCP_LONG_POP_USERNAME: This rule fires when a very long POP3 user name is observed.

  ► TCP_MAX_POP_USERNAME defines the maximum allowable length of a POP3 user name without triggering this rule.

- TCP_LONG_POP_PASSWORD: This rule fires when a very long password is sent to the POP3 port of a machine.

  ► TCP_MAX_POP_PASSWORD defines the maximum allowable length of a POP3 password without triggering this rule.

- TCP_BACK_ORIFICE: This rule looks for TCP traffic from a nonprivileged port to the default port of Back Orifice on a Windows machine. Back Orifice provides a backdoor through which an attacker can access machines.

- TCP_NETBUS_INSTALLED: This rule monitors TCP traffic from certain ports (which are known to be associated with the NetBus tool) of an internal Windows machine. It fires when a "NETBUS" string in the TCP data is detected.

- TCP_NETBUS_PROBE: This rule looks for TCP traffic from a nonprivileged port to a port that is known to be associated with the NetBus tool on a Windows machine.

- TCP_SUSPICIOUS_PORT_PROBE: This is a general rule for detecting port scanning activities.

- TCP_SYNFLOOD: This rule detects the TCP SYN flood attack. The SYN flood attack can cause denial of service by preventing other machines from establishing TCP connections to the target machine.

- TCP_UNACCEPTABLE_PORT_FLAG_SYN: Alert on seeing a packet with the SYN flag set (but not the ACK flag), going to a listed port.

  ► TCP_UNACCEPTABLE_SYN_PORTS is the list of destination ports for which to raise alerts about SYN packets.

- TCP_UNACCEPTABLE_PORT_FLAG_SYN_ACK: Alert on seeing a packet with both the SYN and ACK flags set, coming *from* a listed port.

  ► TCP_UNACCEPTABLE_SYN_ACK_PORTS is the list of source ports for which to raise alerts about SYN-ACK packets.

- TCP_UNACCEPTABLE_PORT_FLAG_ACK: Alert on seeing a packet with the ACK flag set (but not the SYN flag), going to a listed port.

  ► TCP_UNACCEPTABLE_ACK_PORTS is the list of destination ports for which to raise alerts about ACK packets.

- TCP_UNACCEPTABLE_PORT_PATTERN: Alert on seeing a packet going to a listed port, and which also contains a listed pattern in its data portion.

  ► TCP_UNACCEPTABLE_PORTS is the list of destination ports for which to raise alerts about packets with a listed data content.

  ► TCP_UNACCEPTABLE_PATTERNS is the list of data patterns (substrings) to look for in the TCP data portion of the packet.

3.3.2 eXpert-UDP rulesets

- UDP_TEARDROP: This rule detects overlapping UDP datagram fragments, which are a characteristic of the Teardrop attack. The Teardrop attack can cause denial of service.

  ► UDP_TEARDROP_WINDOW defines the size of the time window (in seconds) within which Teardrop attacks are aggregated.

- UDP_LAND: This rule detects UDP datagrams whose source IP address is equal to its destination IP address and its source port number is equal to its destination port number. This attack can cause denial of service.

- UDP_DNS_Poison: This rule fires when there is a mismatch between a domain name queried by an internal host and the domain name for which an external name server provides DNS data. This is a form of DNS cache-poisoning attacks. If successful, it can cause authentication to fail or denial of service.

  ► UDP_DNS_EXPIRE defines the size of the time window (in seconds) during which this rule keeps track of outstanding UDP DNS queries.

- UDP_DNS_Hinfo: This rule fires when an external host sends an HINFO (host information) query. This may indicate intelligence gathering activity.

- UDP_DNS_Hostname_Overflow: This rule detects DNS replies containing very long host names sent from an external host to an internal name server. This may indicate a buffer overflow attack on the name server.

  ► UDP_MAX_DNS_QUES_LEN defines the maximum allowable length of a host name in a DNS reply.

- UDP_DNS_A_Length_Overflow: This rule fires when an external host sends a DNS reply that contains a length field larger than 4. An attacker may obtain root privileges using this attack.

- UDP_NFS_File_Handle_Guess: This rules detects excessive NFS errors, a symptom of NFS file handle guessing attack. If successful, an attacker may access and modify sensitive files.

  ► UDP_NFS_ERROR_EXPIRE defines the size of the time window (in seconds) during which this rule keeps track of previous NFS errors.

  ► UDP_MAX_NFS_GUESSES defines the number of NFS errors required to trigger this rule.

- UDP_NFS_Device_Creation: This rule fires when a user tries to create a device on an NFS server.

- UDP_Echo_Amplifier: This rule detects heavy traffic involving the ECHO service. This may be due to a denial-of-service attack that involves the CHARGEN and the ECHO services.

  ► UDP_AMPLIFIER_ALERT_TIMEOUT defines the size of the time window (in seconds) during which this rule remembers datagrams involving the UDP ECHO service.

- UDP_SNMP_Guess: This rule fires when the number of SNMP password guessing attempts from an external host to an internal host within a certain time period exceeds a certain threshold.

  ► UDP_MAX_SNMP_GUESSES defines the threshold on the number of failed SNMP guesses for firing this rule.

  ► UDP_SNMP_GUESS_ALERT_TIMEOUT defines the size of the time window (in seconds) within which SNMP activities between two machines are aggregated.

- UDP_SNMP_Get: This rule detects attempts to use a public SNMP password by an external host to gain access to an internal host.

  ► UDP_SNMP_GET_ALERT_TIMEOUT defines the size of the time window (in seconds) within which this rule aggregates multiple SNMP GET alerts.

- UDP_Syslog: This rule looks for an external host accessing port 514 of an internal host. When Solaris syslogd (a Unix system logging daemon) receives an external message, it attempts to perform a DNS query on the source IP. If this IP does not match a valid DNS record, the syslogd may crash.

▶ UDP_SYSLOG_WINDOW defines the size of the time window (in seconds) during which this rule aggregates syslogd denial-of-service attacks involving the same source and the same destination hosts.

- UDP_Port_Sweep: This rule fires when the number of distinct ports of a host that are probed within a certain time period exceeds a certain threshold.

  ▶ UDP_UPORT_SWEEP_THRESHOLD defines the number of distinct ports belonging to the same host that are hit in order to trigger this rule.

  ▶ UDP_PORTSWEEP_WINDOW defines the size of the time window (in seconds) within which port sweeps targeting a certain host are aggregated.

- UDP_Adress_Sweep: This rule fires when the number of distinct IP addresses scanned by a single host within a certain time period exceeds a certain threshold.

  ▶ UDP_UADDR_SWEEP_THRESHOLD defines the number of distinct addresses scanned by a single source before this rule will fire.

- UDP_Port_Flood: This rule fires when the number of datagrams sent to a single port of a host within a certain time period exceeds a certain threshold. This may indicate a denial-of-service attack. Certain ports that usually have a lot of traffic can be excluded from this analysis.

  ▶ UDP_SINGLE_UPORT_FLOOD_THRESHOLD defines the number of times a single port is hit within a certain time period before a port flood is declared.

  ▶ UDP_HIGH_TRAFFIC_PORTS lists ports that are known to have a lot of UDP traffic, and therefore should be ignored by this rule.

3.3.3 eXpert-ICMP rulesets

- ICMP_POD_RECONSTRUCTION_ATTACK: This rule detects the ping-of-death attack by looking for ICMP ECHO packets that are longer than 65535 bytes. A ping-of-death attack can cause the target machine to crash.

  ▶ ICMP_POD_REPORT_TIMEOUT defines the amount of time (in seconds) after the last ping-of-death attack from a host to another before the expert reports the aggregated number of ping-of-death attack instances between these two machines.

- ICMP_BROADCAST_ECHO: When an ICMP ECHO packet sending from an external IP address to a broadcast IP address is observed, this rule will fire. This broadcast ECHO attack can cause many packets to be sent to a victim machine to achieve denial of service.

- ICMP_RECONSTRUCTION_ERROR: This rule fires when abnormal ICMP packets (e.g., very large packets) or malformed ICMP packets (e.g., having fragments that overlap) are observed. The following types of ICMP packets are excluded from this rule: ECHO, ECHO REPLY, and DESTINATION UNREACHABLE.

- ICMP_EXCESSIVE_ECHOS_SINGLE_CLIENT: This rule fires when the number of ICMP ECHO packets sent by a single host within a certain time period exceeds a certain threshold.

  ► ICMP_MAX_ECHOS_SINGLE_CLIENT defines the number of ICMP ECHO packets having the same source IP address within a certain time period needed to trigger this rule.

- ICMP_SMURF: This rule fires when the number of unsolicited ICMP ECHO REPLY packets (i.e., ECHO REPLY packets that do not have a matching ECHO packet sent by the target host) within a certain time period exceeds a certain threshold. This SMURF attack can cause network congestion.

  ► ICMP_SMURF_COUNT_THRESHOLD defines the total number of unsolicited ECHO REPLY packets observed within a certain time period to trigger this rule.

- ICMP_EXCESSIVE_ECHOS_ALL_CLIENTS: When the number of ICMP ECHO packets observed within a certain time period exceeds a certain threshold, this rule will fire.

  ► ICMP_MAX_ECHOS_ALL_CLIENT defines the number of ECHO packets in a certain time period needed to trigger this rule.

- ICMP_UNREACHABLE_STORM: This rule fires when the number of DESTINATION UNREACHABLE packets observed within a certain time period exceeds a certain threshold.

  ► ICMP_DEST_UNREACHABLE_THRESHOLD defines the total number of DESTINATION UNREACHABLE packets in a certain time period needed to trigger this rule.

- ICMP_SOURCE_QUENCH: This rule fires when the number of SOURCE QUENCH packets going to an internal host within a certain time period exceeds a certain threshold. When a host receives an ICMP SOURCE QUENCH packets, it will slow down sending packets. Although deprecated (c.f. RFC 1812), ICMP SOURCE QUENCH packets are still being used. Thus legitimate SOURCE QUENCH packets may be observed. On the other hand, ICMP SOURCE QUENCH packets can potentially be used to perform denial/degradation of service.

  ► ICMP_MAX_SRC_QUENCH defines the number of SOURCE QUENCH packets needed to trigger ICMP_SOURCE_QUENCH.

- ICMP_REDIRECT: This rule fires when the number of ICMP REDIRECT packets going to an internal host within a certain time period exceeds a certain threshold. ICMP REDIRECT packets are used to notify a host that it is using a nonoptimal route to send a packet. There is a legitimate use for ICMP REDIRECT packets, but this should happen infrequently. Using ICMP REDIRECT packets, an attacker could subvert a host by masquerading as a trusted host, or cause denial of service to a host by tricking it to use a wrong route.

  ▶ ICMP_MAX_REDIRECT defines the number of ICMP REDIRECT packets needed to trigger ICMP_REDIRECT.

- ICMP_TIME_EXCEEDED: This rule fires when the number of ICMP TIME EXCEEDED packets going to an internal host within a certain time period exceeds a certain threshold. Excessive TIME EXCEEDED packets may indicate ongoing network reconnaissance activities (e.g., using traceroute).

  ▶ ICMP_MAX_TIME_EXCEEDED defines the number of ICMP TIME EXCEEDED packets needed to trigger ICMP_TIME_EXCEEDED.

- ICMP_MISC_FLOOD: This rule fires when the number of uncommon ICMP packets (i.e., ICMP packets that are not of the following types: ECHO, ECHO REPLY, DESTINATION UNREACHABLE, SOURCE QUENCH, TIME EXCEEDED, and REDIRECT) observed in a certain time period exceeds a certain threshold.

  ▶ ICMP_MAX_OTHER_ICMP defines the number of uncommon ICMP packets needed to trigger ICMP_MISC_FLOOD.

3.4 Application protocols

For the application protocols FTP, HTTP, SMTP, and "SESSION" (rlogin and telnet), *etcpgen* performs extensive reconstruction and bookkeeping to be able to provide the analysis engines with messages that are on a higher abstraction level than the underlying transport protocols represent. This has enabled us to develop intelligent rulesets that can detect forms of misuse that it is simply not possible to determine from single packets outside the session context.

3.4.1 eXpert-FTP rulesets

- FTP_DESTRUCTIVE_CMD_ATTEMPT: When an anonymous user attempts to issue a "destructive" FTP command *unsuccessfully*, this rule will generate an incident report. Destructive FTP commands refer to the ones that can modify the file system of an FTP server (e.g., STOR).

  ▶ FTP_DESTRUCTIVE_CMD specifies a list of FTP commands that can modify the file system of an FTP server.

- FTP_DESTRUCTIVE_CMD_SUCCESS: This rule detects incidents in which an anonymous user issues a destructive FTP command *successfully.*

  ► FTP_DESTRUCTIVE_CMD specifies a list of FTP commands that can modify the file system of an FTP server.

- FTP_RESERVED_NAME: This rule fires when an attempt to start an FTP session using a "reserved" account is detected. These reserved accounts are not normally used by an ordinary user, and they are usually well-known system accounts (e.g., bin).

  ► FTP_RESERVED_ACCOUNT is a list of FTP account names that are not normally used by an ordinary FTP user.

- FTP_SENSITIVE_FILE_RETR: If a retrieve command (RETR) with an argument that corresponds to a sensitive file is detected, this rule will fire. Retrieving these files may indicate an attempt to steal security-related information.

  ► FTP_SENSITIVE_FILE is a list of names corresponding to sensitive system or user files.

- FTP_SITE_EXEC: This rule fires when an anonymous user issues a SITE command with EXEC as an argument. A successful SITE EXEC attack can lead to root compromise on the FTP server.

- FTP_CWD_PROBES: This rule fires when an anonymous user uses the CWD command to access files in sensitive directories (e.g., /usr). Accessing these directories by an anonymous user may violate a access control security policy.

  ► FTP_SENSITIVE_DIR contains a list of directories that may contain sensitive files and thus should not be accessed by anonymous users.

- FTP_NLST_DENIAL: This rule fires when an NLST command is issued with "../*/../*/../*" as an argument. This attack may cause denial of service to the FTP server.

- FTP_BAD_LOGIN: This rule generates an incident report when a certain number of failed FTP login events are detected within a certain period of time. Repeated failed login events may indicate a password guessing attack.

  ► FTP_MAX_BU_LIFE defines the size of the time window (in seconds) during which a failed login event is considered by this rule.

  ► FTP_MAX_FAILED_THRESHOLD defines the number of failed login events required to trigger this rule.

- FTP_CORE_ATTACK: A successful FTP core dump attack can enable a legitimate user to obtain the contents of a Unix shadow password file. This attack involves several steps: (1) logon via FTP with your regular username and password; (2) cd /tmp; (3) user root <incorrect password>; (4) quote pasv; (5) get the core file, which contains the passwords, from /tmp.

▶ FTP_MAX_BU_LIFE defines the size of the time window (in seconds) during which a failed login event is considered by this rule.

- FTP_BUFF_OVERFLOW: This rule detects very long FTP command arguments that contain certain suspicious patterns. They may indicate a buffer overflow attack.

  ▶ FTP_MAX_FTPSTR_LEN defines the maximum allowable length of FTP command arguments.

  ▶ FTP_BO_PATTERN contains patterns in FTP command arguments (in addition to a list of patterns included in the expert) that correspond to buffer overflow attacks.

- FTP_BOUNCE: This rule detects an FTP server being used as a third-party bounce point to attack other sites by means of the PORT command. The FTP bounce attack enables an attacker to probe other machines (while hiding the origin of the attack) and to circumvent network access control (e.g., bypassing packet filters and violating export control).

3.4.2 eXpert-HTTP rulesets

The heuristics marked with (*) fire only when a request contains any of the arguments in HTTP_NASTY_FILE_ARGS_DETECT but not any of the arguments in HTTP_NASTY_FILE_ARGS_EXCEPT. The heuristics marked with (+) fire only when a request refers to a CGI directory in HTTP_CGI_LOCATION_DETECT but not in HTTP_CGI_LOCATION_EXCEPT.

- HTTP_TOO_LARGE_REJECT: A server reply with status code 413 (Request Entity Too Large) or 414 (Request-URI Too Large) is observed. It could indicate attempts to run buffer overflow attacks against the server, but could also be caused by misconfigured scripts or clients.

- HTTP_SUSPICIOUS_ENCODING: This rule fires when suspicious encoding is found in the URI, such as if ordinary 7-bit ASCII letters or digits are hex-encoded (e.g., %70 for 'p'), nonstandard spaces are used, or encoded NULL characters are included. The only reason for such encoding would be to try to hide something (e.g., to elude a string-matching IDS).

- HTTP_SECRET_ACCESS: The URI contains one of the paths specified in the HTTP_SECRET_DIRS list.

  ▶ HTTP_SECRET_DIRS contains a list of directories that should not be accessed through HTTP.

- HTTP_APACHE2: The Apache2 attack is a denial-of-service attack against an Apache Web server where a client sends a request with many HTTP headers. If the server receives many of these requests it will slow down, and may eventually crash.

  ▶ HTTP_APACHE2_THRESHOLD is the number of headers in a request that will trigger this ruleset.

  ▶ HTTP_APACHE2_REPORT_TIMEOUT is the time window (in seconds) within which APACHE2 alerts will be aggregated.

- HTTP_PHF: This rule detects HTTP data that refer to a file whose name includes "phf" that resides in a CGI script directory. The PHF attack could enable arbitrary commands to be executed with the privileges of the Web server. (*,+)

- HTTP_NPH: This rule detects HTTP data referring to a file whose name includes "nph-" that resides in a CGI script directory. An attacker may use the NPH attack to read files that are stored in the Web server. (*,+)

- HTTP_CAMPAS: This rule detects HTTP data referring to a file whose name includes "campas?" that resides in a CGI script directory. A successful CAMPAS attack enables the attacker to execute arbitrary commands with the Web server's privileges. (*,+)

- HTTP_GLIMPSE: This rule detects HTTP data referring to a file whose name includes "glimpse" that resides in a CGI script directory. A successful GLIMPSE attack enables the attacker to execute arbitrary commands with the Web server's privileges. (*,+)

- HTTP_NEWDSN: This rule looks for HTTP data containing the strings "newdsn.exe" and "driver=". This vulnerability of Microsoft IIS enables an attacker to create a Microsoft Access Database file with any file extension.

- HTTP_BACK: This rule fires if HTTP data containing 25 consecutive slash characters ("/") is observed. Performing this attack multiple times may cause denial of service to the HTTP server.

- HTTP_CRASH_IIS: This rule detects HTTP data containing the strings "GET" and "../..". This attack could crash Microsoft IIS.

- HTTP_NOVELL_CONVERT: This rule detects HTTP data containing the string "convert.bas?../". This attack, if successful, enables an attacker to browse the file system with the Web server's privileges.

- HTTP_WEBGAIS: This rule detects HTTP data containing the strings "webgais" and "query=\"'. This WebGais vulnerability could provide remote execution capability to an attacker. (+)

- HTTP_CGI_NASTY_PROBE: This is a general rule for CGI-bin programs that are potentially vulnerable. To fire this rule, the name of a vulnerable CGI program and an argument in HTTP_NASTY_FILE_ARGS_DETECT must be observed in HTTP data. (+)

  ▶ HTTP_CGI_PROGS_W_NASTY_DETECT is a list of CGI program names that, when found with a suspicious argument, may indicate a probe or an attack exploiting vulnerable CGI programs.

- HTTP_CGI_ALONE_PROBE: This is a general rule for CGI-bin programs that are potentially vulnerable. (+)

  ▶ HTTP_CGI_PROGS_ALONE_DETECT is a list of CGI program names that, when found in HTTP data, may indicate a probe or an attack exploiting vulnerable CGI programs. (+)

- HTTP_NASTY_DIR_ARGS: This rule looks for an HTTP "GET" request for a file residing in a directory that is normally not accessed through HTTP.

  ▶ HTTP_NASTY_DIR_ARGS_DETECT contains a list of patterns that correspond to directory paths not normally accessed through HTTP.

  ▶ HTTP_NASTY_DIR_ARGS_EXCEPT contains a list of patterns that correspond to directory paths. Moreover, it is safe for HTTP requests to access these directories.

- HTTP_NASTY_FILE_ARGS: This is a general rule that detects HTTP requests carrying a suspicious file argument. (*)

- HTTP_NASTY_PROGRAM_ARGS: This is a general rule that detects HTTP requests carrying a suspicious program argument.

  ▶ HTTP_NASTY_PROGRAM_ARGS_DETECT is a list of filename patterns that, when found in HTTP data, could indicate attempts to run programs chosen by an adversary on the HTTP server.

  ▶ HTTP_NASTY_PROGRAM_ARGS_EXCEPT contains a list of filename patterns that represent an exception to this rule.

- HTTP_URI_BUFFER_OVERFLOW: Raise an alert when a URI longer than a specified length is observed, as this could indicate a buffer overflow attack. This rule is *disabled by default*, because of the risk of false positives.

  ▶ HTTP_URI_OVERFLOW_LEN is the threshold URI length for overflow alerts.

- HTTP_HEADERS_BUFFER_OVERFLOW: Raise an alert when the combined length of all the HTTP headers of a request exceeds a threshold value, as this could indicate a buffer overflow attack. This rule is *disabled by default*, because of the risk of false positives.

  ▶ HTTP_HEADERS_OVERFLOW_LEN is the threshold header length for overflow alerts.

3.4.3 eXpert-SMTP rulesets

- SMTP_SUSPICIOUS_CMD_NAME: This rule detects SMTP commands that are not supported. Attacks that use SMTP debugging commands such as DEBUG and SHOWQ, the WIZ command (which corresponds to a vulnerability in old versions of sendmail), or insecure SMTP commands (e.g., TURN) can be detected by this rule.

  ► SMTP_EXTENDED_CMD_NAME contains a list of SMTP commands that correspond to SMTP extensions supported by the monitored site.

- SMTP_LONG_CLIENT_CMD_LINE: This rule detects SMTP command lines that are longer than a certain threshold and correspond to a supported SMTP command. Many buffer overflow attacks have the characteristic that long command lines are used.

  ► SMTP_MAX_CLIENT_CMD_LINE_LEN is the maximum length of a command line without triggering this rule.

  ► SMTP_EXTENDED_CMD_NAME lists the extended SMTP commands for which this rule will perform the length check. This rule will check the length of command lines that have a standard SMTP command name.

- SMTP_BAD_PATH: This rule detects MAIL/RCPT command arguments that contain characters/patterns not normally used. For example, some SMTP exploits use mail paths that contain "|" (i.e., the pipe character).

  ► SMTP_SUSPICIOUS_PATH_PATTERN contains a list of suspicious patterns to look for in MAIL/RCPT command arguments.

- SMTP_ALIAS_ATTACK: This rule detects attacks that use certain suspicious mail aliases as RCPT command arguments. For example, attacks that use the "uudecode" mail alias can be detected.

  ► SMTP_SUSPICIOUS_RCPT_MAIL_ALIAS contains a list of mail aliases that appear as RCPT command arguments of SMTP attacks.

- SMTP_MAIL_RELAY: This rule checks for multiple "@" characters and the "%" character in a RCPT command line. Their presence may indicate that an attacker attempts to use an SMTP server as a relay to conduct mail spamming. Disable this rule if your server supports third party mail relaying.

- SMTP_EXT_SRC_EXT_DEST: This rule detects mail transmission involving an external sender (based on the sender's IP address) and an external recipient (based on the recipient's domain name). This activity may indicate mail spamming. Disable this rule if your server(s) supports third-party mail relaying, or if the monitored mail server(s) is allowed to serve clients with an address not in the user-configurable local network address list. Disable this rule when analyzing data collected from another site, or reconfigure SMTP_INTERNAL_DOMAIN_LIST and the local network address list appropriately.

▶ SMTP_INTERNAL_DOMAIN_LIST is a list of domain names used to determine whether an e-mail address should be treated as internal. Because this list is sitespecific, it has no default value. Thus this list *must* be configured properly before enabling this rule.

- SMTP_EXPN_PROBE: This rule detects EXPN commands that involve suspicious mail aliases, especially those that may potentially expand to many e-mail addresses (e.g., "all"), and those that test the presence of attack-related aliases (e.g., "decode"). This probing activity may indicate an attempt to gather intelligence before an attack is launched.

  ▶ SMTP_SUSPICIOUS_EXPN_MAIL_ALIAS is a list of mail aliases that correspond to EXPN command arguments of SMTP probing attacks.

- SMTP_WORM: This rule looks for certain suspicious patterns in mail messages. The presence of these patterns may indicate an e-mail worm/virus.

  ▶ SMTP_WORM_PATTERN contains patterns found in the mail body of an e-mail worm/virus.

- SMTP_LONG_MIME_FILENAME: This rule detects long file names used for a file attachment in a MIME (Multipurpose Internet Mail Extensions) message. Having very long file names for MIME attachments may indicate a buffer overflow attack.

  ▶ SMTP_MAX_MIME_FILENAME_LEN denotes the maximum allowable length for the name of a MIME attachment without triggering the rule.

- SMTP_SENDMAIL_BO: This rule examines the length and the content of SMTP commands to detect a sendmail buffer overflow attack.

  ▶ SMTP_SENDMAIL_BO_PATTERN contains additional patterns in the mail message body that may indicate sendmail buffer overflow attacks.

- SMTP_MAILBOMB: When there are too many mails sent by an SMTP client to an SMTP server, this rule will declare a mail bomb attack. A mail bomb can overflow a mail queue and can potentially cause system failure.

  ▶ SMTP_MAX_MAILBOMB_THRESHOLD denotes the maximum allowable number of mail transmissions sent from a certain client to a certain server without triggering this rule.

  ▶ SMTP_MAX_TRANSFACT_LIFE denotes the amount of time (in seconds) a single mail transmission will be remembered by this rule.

  ▶ SMTP_MAX_TRANSCOUNT_LIFE denotes the amount of time (in seconds) no more activity is observed between an SMTP client and an SMTP server before this rule "forgets" (i.e., restarts the aggregation process of) the mail transmissions sent between them.

- SMTP_SENDER_EVENT_AGGREGATION: This rule aggregates SMTP events that correspond to a client, and it must be enabled if one or more

of the following rules are enabled:
SMTP_MANY_FAILED_VRFY, SMTP_MANY_FAILED_EXPN, SMTP_MANY_ETRN, and SMTP_MANY_RCPT.

▶ SMTP_MAX_SENDER_AGGREGATES_LIFE specifies the amount of time (in seconds) no more activity is observed from an SMTP client before this rules "forgets" the client's prior activities.

- SMTP_MANY_FAILED_VRFY: This rule fires when the number of failed VRFY commands in a mail transmission session exceeds a certain threshold. Failed VRFY commands may indicate intelligence gathering.

  ▶ SMTP_MAX_FAILED_VRFY denotes the maximum allowable number of failed VRFY in a mail transmission session without triggering the rule.

- SMTP_MANY_FAILED_EXPN: This rule fires when the number of failed EXPN commands in a mail transmission session exceeds a certain threshold. Failed EXPN commands may indicate intelligence gathering.

  ▶ SMTP_MAX_FAILED_EXPN denotes the maximum allowable number of failed EXPN in a mail transmission session without triggering the rule.

- SMTP_MANY_ETRN: This rule is triggered when the number of ETRN commands in a mail transmission session exceeds a certain threshold. In some SMTP implementations (e.g., sendmail 8.9.1), an SMTP server forks and sleeps for seconds when an ETRN command is received. Enough ETRN requests may exhaust the resources of an SMTP server to cause denial-of-service or server reboot.

  ▶ SMTP_MAX_ETRN denotes the maximum allowable number of ETRN in a mail transmission session without triggering the rule.

- SMTP_MANY_RCPT: This rule is triggered when the number of recipients specified in a mail transmission session exceeds a certain threshold. A large number of recipients may indicate mail spamming or a denial of service attack.

  ▶ SMTP_MAX_RCPT denotes the maximum allowable number of recipients in a mail transmission session without triggering the rule.

3.4.4 eXpert-SESSION rulesets

- SESSION_External_Rlogin_Request: This rule detects the use of r* commands (e.g., rlogin, rsh, and rexec) from an external host. Because these services use an insecure user authentication mechanism, accessing them from an external host should be disallowed.

- SESSION_Xterm_BO: This rule detects attempts to exploit a known xterm buffer overflow attack.

- SESSION_Unexpected_Root_Transition: This rule fires when a user becomes root without using the su command. This suspicious event may be a symptom of a successful attack.

- SESSION_SU_By_NonAdmin: If a user attempts to obtain root privileges using the Unix su command, and that user is not authorized to do so, this rule will fire.

  ▶ ADMINISTRATIVE_USER_LIST contains the list of users that are authorized to obtain root privileges.

- SESSION_PW_OverWrite: This rule detects attempts to overwrite password files (i.e., passwd and shadow files in Unix systems) or to append new entries to them. This attack may enable an attacker to add new accounts that have known or no passwords, or cause denial of service to existing users.

- SESSION_RH_OverWrite: This rule detects attempts to overwrite or to append to .rhosts files, which specify trust relationships between a user account and other hosts or accounts. If successful, an attacker may log on to the victim's account without knowing the password.

- SESSION_BIN_Overwrite: This rule detects attempts to tamper with the system files in the bin directories of a Unix system. (e.g., system binaries may be replaced or removed).

- SESSION_LOG_OverWrite: This rule detects attempts to tamper with the system log files in the /var/log directory of a Unix system. This kind of activity may indicate that an attacker tries to destroy the evidence of an attack.

- SESSION_Suspicious_Link: This rule fires when symbolic links are created in the /tmp directory.

- SESSION_Suspicious_Argument: This rule looks for certain suspicious keywords in data traffic.

- SESSION_Suspicious_Setuid: This rule fires when a dot file (i.e., the name of a file that starts with ".") is made setuid or when a setuid file is created by an ordinary user.

- SESSION_NTSunKill: This rule looks for four or more consecutive control-D characters in session traffic, which may be used by an attacker to cause denial of service.

- SESSION_Remote_Privileged_Login: This rule detects a login event that involves a reserved account from an external host.

  ▶ RESTRICTED_ACCOUNTS_LIST contains a list of account names that should not be used by a remote login.

- SESSION_MAX_Failed_Logins: This rule fires when the number of failed login events within a certain time period exceeds a certain threshold. This may indicate a password guessing attempt.

  ▶ MAX_FAILED_LOGINS defines the number of failed login events needed to trigger this rule.

  ▶ FAILED_LOGIN_WINDOW defines the size of the time window during which we aggregate failed login events for this rule.

- SESSION_Dictionary_Attack: This rule fires when the number of failed login events within a certain time period exceeds a certain threshold. This may indicate a more elaborate password guessing attempt.

  ▶ DICTIONARY_COUNT defines the number of failed login events observed before this rule declares a dictionary attack.

  ▶ FAILED_LOGIN_WINDOW defines the size of the time window during which we aggregate failed login events for this rule.

3.5 Related work

The first network-based IDS, using data intercepted or "sniffed" from a broadcast Ethernet network, was NSM from UC Davis [22]. Because of the popularity of Ethernet and the ability to monitor many hosts with a single sensor, network-based IDSs became very popular, in the research community [40], the commercial market [25], and the open-source community with Snort [50] and others.

Snort [50] is an example of a simple single-packet analysis mechanism, although there are some add-on extensions to Snort that perform more complex analysis. An example of a tool that performs analysis on the advanced multi-event level of EMERALD *eXpert-Net* is NetSTAT from UCSB [60].

3.6 Conclusions

We have presented a software package for network-based intrusion detection, both on a transport-protocol level and on the application-protocol level. By performing reconstruction of transactions and sessions, it can detect many security problems that would otherwise not be possible to detect solely from network surveillance.

Chapter 4

Host-based Detection

This chapter investigates the possibilities and limitations of host-located audit-based data analysis for intrusion detection and response. In particular, we describe EMERALD *eXpert-BSM*[33], a real-time intrusion detection solution for the Sun Solaris Operating System Environment.

4.1 Introduction

When research on intrusion detection was initiated in the early 1980s, the problem was often referred to as automated audit-trail analysis. In theory, auditing is an important security service that both establishes accountability for users and aids in damage assessment once an abuse is discovered. Unfortunately, in practice the volumes of data that tend to be produced by audit services are such that any security violation recorded within the audit trail is often secure from discovery as well. The increasing speed and complexity of modern computing environments has increased the volumes of audit data that can be produced.

The Solaris Basic Security Module (BSM) [56] is one example of an auditing facility that can provide detailed records about system events. However, for system operators lacking intelligent analysis tools, there are two dominant strategies that emerge in using the audit facility:

1. Turn on auditing for all or most event types, and have a careful scheme in place for copying the large amounts of audit data to secondary storage for its potential use later in forensic analysis.

2. Do not perform auditing at all.

Neither approach utilizes the full potential of auditing facilities as an important contributor to a system's operational security.

The EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) environment is a distributed scalable tool suite for tracking malicious activity through and across large networks [46]. EMERALD introduces a highly distributed, building-block approach to network surveillance, attack isolation, and automated response. A central concept of EMERALD is its distributed, lightweight *monitors*, diverse with respect both to the monitored event streams and to analysis techniques. *eXpert-BSM* represents one example of an EMERALD monitor that can stand alone as an important host protection service, and can also be easily configured to fit into a distributed framework of surveillance, correlation, and response.

eXpert-BSM is a security service for isolating misuse and other security-relevant warning indicators from the Sun Solaris audit facility. Initial development of *eXpert-BSM* began in 1998 and has continued to the present. This chapter describes the design and features of *eXpert-BSM* and how it fills a vital function in security coverage not provided by network intrusion detection services. Section 4.2 discusses the complementary nature of host audit trail analysis and network traffic monitoring. Section 4.3 summarizes the *eXpert-BSM* attack coverage. Section 4.4 presents the *eXpert-BSM* capabilities and unique features while Section 4.5 discusses deployment experiences and performance characteristics. Section 4.6 discusses related work in the area of host-based security analysis.

4.2 Audit data vs. network traffic

An intrusion detection system (IDS) analyzes an event stream in an attempt to categorize the events as normal or intrusive. The first IDSs proposed and developed in the early 1980s were host based, analyzing the audit trails of mainframe computers in search of anomalies and signs of malicious activity. When later applied to networked environments, the dominant architecture was centralized collection and analysis of raw audit data from multiple hosts. The first network-based IDS, using data "sniffed" from a broadcast Ethernet network, was NSM from UC Davis [22]. The network-based trend that followed has been so strong in commercial and free IDSs that many people equate intrusion detection with network traffic analysis.

In this chapter, we somewhat narrowly use the term *host-based* to refer to a monitor that analyzes audit data from the operating system kernel. In referring to host-based intrusion detection, others have included any form of analysis that is focused on the protection of a single host. For example, some IDS developers have proposed placing a network event collector and analyzer locally on every host, observing traffic involving only that host. That would not fit into the definition of host-based analysis as used in this chapter. Accordingly, *network-based* analyses are defined here to involve the analysis of network traffic data, wherever the monitor is located.

Major functional separation between host- versus network-based analyses arises from the content of the data streams being analyzed. Audit-based analyses provide an exceptional degree of insight into the internal operations of processes executing within the host. From the audit-trail vantage point, one can examine all access control decisions occurring between the kernel and user processes, profile normality in process activity, and compare user actions against their expected roles within the system.

Surveillance through network traffic analysis allows a system to view the network communications across multiple hosts. In broadcast networks, a single sensor can provide analysis coverage over an entire local area network (LAN). Both host- and network-based surveillance are important and complementary. Each has its place in the arsenal of INFOSEC devices being made available to supplement the need for computer and network security. However, each approach has its respective weaknesses.

4.2.1 Network-based IDS limitations

A fundamental limitation to network analysis is that not all forms of misuse will necessarily generate network traffic. Further, not all misuse activity that results in network traffic will provide sufficient information to isolate the misuse. Examples of such information include the true full local pathname of a file retrieved through HTTP, or the user ID under which a particular service daemon executes. This is also a problem with buffer overflows and other well-known malicious attacks that are performed from the console or over an encrypted channel.

Application-layer encryption of network traffic is becoming more common and user transparent thanks to technology such as SSL-enabled Web browsers and Secure Shell (ssh). The same is true for lower-layer encryption through virtual private networks, some of which are based on the IPSEC standard. While this is a positive step forward in communications integrity and the prevention of data theft, it makes network-based intrusion detection more difficult as potentially malicious instructions are also encrypted.

Another problem with network intrusion detection involves the evolution of common network topologies, specifically, the growing popularity of non-broadcast networks. Inserting a network sniffer in the path of all LAN traffic is becoming more challenging. For example, switching technology allows improved network performance by effectively turning a broadcast Ethernet network into a unicast network, hampering sniffing opportunities. Also, if there are multiple possible routes between two communicating hosts, some packets could be routed around the sniffer location.

When intercepting and analyzing the communication between two hosts, it is of paramount importance for correct analysis that the traffic is interpreted equally by the IDS and the receiver. If not, the IDS could be tricked into interpreting traffic as benign while the receiver, making a different interpretation, becomes the victim of an attack. With respect to IP stacks, there are many subtle differences among operating systems that could be used by an attacker to send instructions that appear benign to the IDS, but have malign effects on the victim host [48]. The same holds for application-level interpretation. For instance, Web servers for the Windows platform tend to accept the backslash character as a valid path separator in addition to the forward slash, while servers on Unix platforms do not.

Network traffic analysis is also challenged with the need to provide transaction and session reconstruction, requiring great efficiency in managing state. In many cases, a single packet is not sufficient to correctly identify intrusive behavior. For advanced analysis, the IDS must reconstruct transactions and sessions based on the observed data and therefore keep potentially large amounts of state information for arbitrarily long periods of time. Merely combining the requests and replies across many parallel sessions into transactions can be a complex task for the IDS.

Finally, there is the issue of scalability of a network IDS to large traffic volumes. For line speeds where relatively simple routing decisions have to be made in firmware to be sufficiently fast, the more complicated analysis required by an IDS implemented in software has little chance to keep up.

4.2.2 Host-based IDS limitations

Host-based intrusion detection can avoid most, if not all, of the problems listed above. Thus, it is an important complement to the threat coverage of network-based monitoring. However, host-based monitors also have a set of general problems associated with them.

As with network traffic analysis, host-based analysis is limited by the available content in the event stream. For example, a host-based monitor can fail to observe network-related activity. This illustrates the complementary nature that host analysis shares with network traffic analysis tools. Unfortunately, the use of network-based vulnerability scanners has become a prominent practice in security evaluation procedures, and an evaluator pointing a network scanner against a host equipped with a host-based IDS is often disappointed when the IDS does not react to all elements of the scan. Very severe host attacks readily detectable with host-based analysis are similarly often not recognizable by network IDSs.

Another potential issue with a host-based IDS is its vulnerability to attack once a system has been compromised. When an attacker has taken over the omnipotent super-user account (root, administrator), then, in the absence of automated response, the IDS is itself subject to attack. If the IDS transmits alarm information over the network to another entity, it may be able to report super-user subversion to others before the attacker can stop the IDS.

If a denial-of-service attack brings down the host, the IDS will go down with it. The IDS may be able to raise an alarm about a resource-exhaustion attack in progress, while there could be other attacks that crash the host with only a minimal number of network packets, before the IDS can send out an alarm. The additional load put on the host by the IDS monitor could also be of concern.

4.3 eXpert-BSM knowledge base

Among the first steps toward developing an effective and maintainable misuse detection service is to select a reasoning strategy and knowledge representation structure that is well suited and efficient for this problem domain. In [32], we argue why forward-chaining rule-based systems are highly useful for computer and network intrusion detection. The core of *eXpert-BSM* consists of an inference engine and knowledge base built with the Production-Based Expert System Toolset (P-BEST), a highly optimized forward-chaining rule-based system builder for real-time event analysis.

In the field of expert system analysis, forward-chaining strategies dominate applications that provide prognosis, monitoring, and system control. Generally, forward-reasoning systems excel in expressing logical inferences across multiple events in search of specific event sequences or activity that crosses predefined thresholds of normalcy. *eXpert-BSM*'s P-BEST models can comprehend intrusive behavior that may involve complex/multiple event orderings with elaborate pre- or post-conditions. This allows for a concise rule base, while still being able to recognize wide variation in intrusive activity.

In contrast, a variety of signature-based intrusion detection techniques employ stateless reasoning to isolate single-step malicious activity, such as rudimentary pattern matching. For very high-volume event analysis, stateless predicate reasoning can be quite effective for simple single-packet exploit detection. However, limited expressibility in misuse definitions can lead to inflated rule bases to cover all variations of a known phenomena. Rudimentary pattern matching also fails to cover multievent scenarios.

From 1996 to the present, P-BEST has been employed in the development of nine independent intrusion-detection engines under the EMERALD framework of distributed sensors managed under a correlation hierarchy. P-BEST has shown itself to be an effective real-time transaction processing system, with a pre-compilation library that allows its inference engines and knowledge bases to be easily integrated into large program frameworks. Its language is small and easily extendible, as calls to arbitrary C functions are possible anywhere in the rule structure. Since its inception on Unix, P-BEST has undergone many optimizations (Section 4.5.3).

The P-BEST toolset consists of a rule translator and a library of runtime routines. When using P-BEST, rules and facts are written in the P-BEST production rule specification language. The misuse detection P-BEST compiler, *pbcc*, is then used to translate the P-BEST knowledge specification into a callable expert system library. A full discussion of the P-BEST language definition with examples is provided in [32].

4.3.1 eXpert-BSM attack coverage

The *eXpert-BSM* knowledge base consists of 123 P-BEST rules, which allow *eXpert-BSM* to recognize 46 general forms of misuse or warning indicators of abuse. Initial development of this rule base began in 1998 and has continued to the present. Based on experimental evaluations (see Section 4.5.2) and other input, *eXpert-BSM*'s knowledge base has been refined and extended into an effective suite of intrusion models for identifying, where possible, the broadest forms of activity that indicate abusive or intrusive activity on Unix hosts.

eXpert-BSM excels at detecting when an adversary attempts to violate security on the host, regardless of whether it is an external agent or an insider operating from the console. While attack space coverage is not claimed to be complete, significant effort and experience has been invested in this knowledge base over several years. The majority of the intrusion models attempt to recognize the most general form of misuse by detecting state transitions that represent the underlying compromise of intrusions and other known misuse activity. With respect to attack coverage, these intrusion models are categorized under the following broad areas of host misuse:

Data Theft — involves attempts by non-administrative users to perform read operations on files or devices in a manner considered inconsistent with the system security policy. This includes attempts to access files stored in nonpublic directories that are owned by other users, or to reference files in violation of *eXpert-BSM*'s surveillance policy (Section 4.3.2). The category includes attempts to access root core file contents, a well-known method to gain access to encrypted or even cleartext password content. The category also includes opening network interface devices in promiscuous mode, indicating attempts by non-administrative users to sniff traffic from the network.

System/User Data Manipulation — attempts by non-administrative users to modify system or user data, where *modify* broadly means attempts to alter, create, overwrite, append, remove, or change content or the attributes of file system objects. Coverage in this category includes attempts to modify system files within which security-relevant configuration parameters are stored. This configurable list of files typically includes files in */etc*. The category also includes attempts by anonymous FTP users to modify file system content outside a predefined upload directory, should one exist. Intrusion models that detect attempts to modify user environment files (e.g., *.csrhc*, *.login*, or *.rhosts*) or modify files in violation of *eXpert-BSM*'s surveillance policy (Section 4.3.2) are also included in this category. Last, the category provides comprehensive coverage over attempts to modify system executable binaries and scripts stored in publicly shared binary directories.

Privilege Subversion — provides broad and effective coverage of illegal attempts to subvert root or other administrator authority, either through the illegal changing of one's operating authority, subverting the function of a privileged (setuid) application, or by causing a setuid process to execute an application that is not owned by the setuid program owner or the system. Intrusion models in this category are capable of detecting the three variations of buffer overflow attacks [43, 8] that continue to plague Unix setuid applications and inetd services: exec argument buffer overflows, environment variable overflows, and data-segment overflows. The generality of *eXpert-BSM*'s buffer overflow model was demonstrated when a new such exploit was published for Solaris 8 [14]. Without being updated with specific knowledge about the new attack, *eXpert-BSM* detected and correctly identified this event as privilege subversion through a buffer overflow. Overall, the BSM audit trail provides far superior event content than network traffic from which to identify process subversion on hosts. Inetd service subversion (such as the well-known *sadmind(1)* attack on Solaris [9]) is an example of a data segment buffer overflow, while exec argument and environment variable overflows have been exploited in perhaps more than a dozen setuid applications on Solaris alone (e.g., *eject*, *fdformat*, *ffbconfig*, *passwd*, *ping*, *rdist*, *rlogin*, *ufsrestore*, and *xlock*).

Account Probing and Guessing — identifies repeated attempts to enter a system via authentication services such as rlogin or FTP, or attempts to gain root authority by non-administrative users or from external clients.

Suspicious Network Activity — recognizes various attempts to probe or scan the host, or misuse the host's FTP services to distribute content to other external sites (i.e., FTP warez hosting). While BSM audit trails provide minimal insight into raw network traffic activity, they do allow the monitor to recognize successful connections to TCP-based services and, more important, provide detailed insight into the internal operations of the network server process as it is being used (or potentially misused). *eXpert-BSM* also provides a TCP port scan detection capability on TCP ports that have enabled services on the host.

Asset Distress — identifies operational activity that indicates a current or impending failure or significant degradation of a system asset. The majority of these problems are very difficult, if not impossible, to diagnose via network traffic analysis. This category includes filesystem or process table exhaustion, and also core-dump events by root-owned services. In addition, this category includes detection of malicious service denials, both through remote agents attempting to exhaust process tables via inetd services, and by a Solaris-specific self echo flooding attack by local host processes.

User-specifiable Surveillance — allows the *eXpert-BSM* operator to create site-specific policies on activity that should trigger an immediate alarm. This category includes the ability to recognize operator-defined command arguments considered suspicious for that site and worthy of administrative review. In addition, the operator can specify network ports that should not be accessed by external clients. Examples may include TCP ports 53 (DNS zone transfer), 143 (imapd), or 514 (syslogd). This category also includes the addition of a powerful feature for specifying a site surveillance policy to monitor user accesses to data and executable files (discussed in Section 4.3.2).

Other Security-relevant Events — provides other general security-relevant activity reports worthy of review by security administrators. This includes significant backward movement of the system clock beyond what is normally performed by clock synchronization protocols. This backward time movement is a possible indicator of an attacker attempting to manipulate file or log state to reduce the risk of detection. This category recognizes setuid enabling by non-administrative users, and suspicious symbolic link creation in publicly writable directories. Process execution by reserved accounts that are not intended to run applications (e.g., bin, sys) are also recognized. Finally, this category recognizes attempts to alter the underlying audit configuration, potentially in an effort to flood or starve *eXpert-BSM*.

4.3.2 File surveillance policy specification

eXpert-BSM provides a facility for specifying a surveillance policy over file reads, writes, and executions. Under this policy, the *eXpert-BSM* operator may specify groups of users, files, and directories, and then use these groups to specify surveillance policies regarding file accesses. This allows the operator to easily establish rules for generating immediate notification when users step outside their designated roles in the system.

For example, consider a consultant who is granted access to parts of a file server that also contains company sensitive data to which the consultant should not have access. The operator would, as a first line of defense, set access controls on files and directories to prevent the consultant's access to these sensitive file system areas. However, over time users who work in these areas may fail to continually manage the proper settings on these files and directories, or may create new sensitive files that by default allow the consultant access. The surveillance policy allows the operator to easily detect whenever the consultant accesses, or even attempts to access, files or directories in the sensitive areas of the file system.

There are three distinct components to be specified within an *eXpert-BSM* access policy specification. The first component, the *UserGroups* section, allows the operator to specify groups of users, which are then referenced in the access policy. The names specified under the user groups should be present as valid login names defined within the password file, and user names can appear in multiple lists.

The second section, *FileGroups*, allows the operator to specify a set of files and directories that may be referenced together as a group while enumerating the access policy. Files specified in the file groups should be fully qualified pathnames. The operator can also specify directories, as shown in the example surveillance policy specification in Figure 4.1. Files and directories can appear in multiple lists.

The third section is *Policy*, within which the operator can specify illegal read, write, and execute accesses between users and files. The policy section essentially defines access mode relations among user groups and file groups. For each user group entered in the policy, three possible relations can be specified: *nread*, *nwrite*, and *nexec*. The *nread* mode indicates that users in the associated list are not allowed to read files matching the file lists specified in the bracket clause. Illegal file writes and executions are specified similarly. Figure 4.1 presents an example of an *eXpert-BSM* access policy specification.

```
UserGroups { RegStaff   (user1 user2)
             Management (admin)
             Accounting (acct)
}
FileGroups { Programs ( /bin /usr/bin /usr/local/bin /usr/local/ftp/bin )
             Admtools ( /etc/bin /etc/sbin /usr/sbin /sbin )
             CompanySecrets  ( /secret )
             Payroll  ( /accounting/DBMS/payroll.db )
}
Policy {     RegStaff (
                   nread[CompanySecrets Payroll]
                   nwrite[CompanySecrets Programs Payroll Admtools]
                   nexec[Admtools] )
             Management(
                   nread[]
                   nwrite[Programs Admtools]
                   nexec[] )
             Accounting (
                   nwrite[Programs Admtools]
                   nread[CompanySecrets]
                   nexec[Admtools] )
}
```

Figure 4.1: Example surveillance policy.

In Figure 4.1, there exists a small group of regular staff defined as *user1* and *user2*. There is a management staff, with one manager *admin* and an accounting group consisting of user *acct*. Four file groups are defined. The first file group is the programs group, where programs are defined as being located in */bin, /usr/bin, /usr/local/bin*, and */usr/local/ftp/bin*. An administrative tools group consists of files in */etc/bin, /etc/sbin, /usr/sbin*, and */sbin*. A directory containing company secrets is named */secret*. A payroll file group consists of a file called */accounting/DBMS/payroll.db*.

In Figure 4.1, regular staff members are not allowed to read company secrets or payroll data, as specified by the associated *nread* function. Regular staff may not write to files in the company secrets, programs, payroll, or admin tools, and regular staff may not execute admin tools. If *eXpert-BSM* observes user activity that contradicts this policy, an alert is raised. Members of the management staff are not allowed to modify files in the program or admin tools file groups, but have unrestricted read and execute access over the entire system. Members of the accounting staff are not allowed to modify files in the program or admin file groups, read company secret files, or execute admin tools.

4.4 eXpert-BSM architecture and features

The preceding section discusses the threat coverage provided by the *eXpert-BSM* knowledge base. This section provides an overview of the features and capabilities of the *eXpert-BSM* distribution package, and its management when deployed on multiple hosts in a network.

4.4.1 Preprocessing the Solaris BSM event stream

eXpert-BSM runs on Solaris 2.6, 7, and 8, in 32-bit and 64-bit operating modes. These versions of Solaris have an auditing mechanism known as SunSHIELD Basic Security Module [56], or BSM for short. BSM has its roots in the C2 compatibility package for SunOS 4.x, developed to comply with the TCSEC [57] (Orange Book) requirements.

Before analyzing audit records, *eXpert-BSM*'s event preprocessing service, *ebsmgen*, first transforms the content of each audit record into an internal message structure. These messages include two important synthetic fields, called *synthetic_parentCmd* and *synthetic_parentIP*. Although audit records provide detailed information regarding each system call, they do not identify the command (process image name) under which the system call was invoked. The *synthetic_parentCmd* field tracks this important attribute by observing *exec* calls. Second, although Solaris audit records are structured to include information regarding source IP information for transactions not performed from the console, this information is unreliable across audit event types and OS versions. By tracking the source IP information and always reporting it in *synthetic_parentIP*, *ebsmgen* provides consistently correct IP information for all audit records.

Each message is passed on from the preprocessor to the event handling interface of the expert system, where it is asserted as a fact according to a fact type definition known as a *ptype* in P-BEST [32]. Figure 4.2 shows the ptype definition for audit records, as used in *eXpert-BSM.* Each field in the ptype corresponds to audit token data fields, except for the two synthetic tokens explained above.

Developing tools for analysis of BSM data is not without difficulty. The only tool provided with Solaris for audit data interpretation is *praudit*, which prints audit data in a simple text form. Although Solaris has some library routines for producing binary audit records, there are no routines available for consumers of BSM data. There is no formal grammar specified for BSM to help developers of consumer tools, and an effort to specify such a grammar for the BSM of Solaris 2.6 encountered some difficulties [18]. In Solaris 7 and 8, several new and initially undocumented audit token types were introduced. These are related to the support of 64-bit mode and other new features such as IPv6.

The syntax of audit records and audit tokens is relatively well specified in the documentation, but the semantics of the content is not. This is especially true for audit records generated by applications outside the kernel, such as the *login* program. For developers of BSM audit trail analysis tools, such as the system described in this report, this necessitates empirical studies of large

```
ptype[bsm_event
    human_time:           string,    'Header timestamp as a string.
    header_event_type:    int,       'Header event numerical ID
    header_time:          int,       'Header time as a numeric value.
    header_command:       string,    'Header event ID as a string (event name)
    header_size:          int,       'Header byte count
    msequenceNumber:      int,       'Sequence token number
    path_List:            string,    'Paths from one or several path tokens
    subject_auid:         int,       'Subject audit ID
    subject_euid:         int,       'Subject effective user ID
    subject_ruid:         int,       'Subject real user ID
    subject_pid:          int,       'Subject process ID
    subject_sid:          int,       'Subject audit session ID
    subject_machine_ID:   string,    'Subject machine ID
    in_addr_address:      string,    'In_addr Internet address
    in_addr_hostname:     string,    'In_addr Internet hostname
    attr_uidList:         int,       'Attribute owner UID
    val_List:             int,       'Argument value
    return_return_value:  int,       'Return process value
    return_error_number:  int,       'Return process error
    textList:             string,    'Text strings from one or several text tokens
    exec_args:            string,    'Exec arguments
    exec_env_txt:         string,    'Exec environment
    sock1_sock_type:      int,       'Socket type
    sock1_remote_port:    int,       'Socket remote port
    sock1_remote_iaddr:   string,    'Socket remote IP address
    sock1_local_port:     int,       'Socket local port
    sock1_local_iaddr:    string,    'Socket local IP address
    sock2_sock_type:      int,       'Socket type for second socket token
    sock2_remote_port:    int,       'Socket remote port for second socket token
    synthetic_parentCmd:  string,    'Synthetic parent command
    synthetic_parentIP:   string     'Synthetic parent IP address
]
```

Figure 4.2: The P-BEST ptype for BSM events.

amounts of audit data to understand the semantics of the BSM data stream. Undocumented changes between different versions of Solaris contribute to the difficulty of this task.

4.4.2 Modes of operation

eXpert-BSM requires no reactive probing of the system state, resulting in an IDS that produces identical results in batch and real-time modes. Batch-mode processing allows *eXpert-BSM* operators to process previously archived audit files, typically created by *auditd*. In real-time mode, *eXpert-BSM* is able to analyze audit records as they are produced by the kernel. The *eXpert-BSM* inference engine is packaged together with three additional modules that cooperate to relay BSM records directly from the kernel to the *eXpert-BSM* inference engine via interprocess communication, as illustrated in Figure 4.3. In its real-time operating mode, the *eXpert-BSM* package employs the following modules:

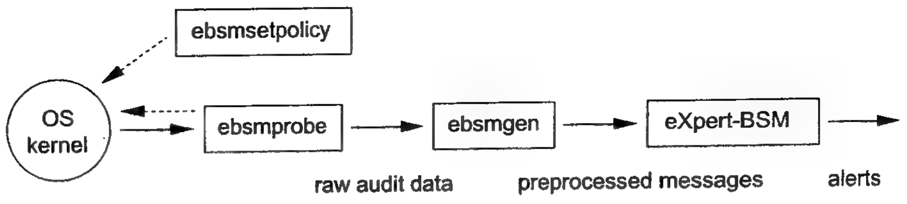


Figure 4.3: The components and data flow in the eXpert-BSM process chain (real-time mode).

ebsmsetpolicy — (real-time mode) is a small setuid to root application that configures the desired audit policy for the kernel, and then terminates immediately.

ebsmprobe — (real-time mode) establishes process-to-process communication between the Solaris kernel and *ebsmgen*. *ebsmprobe* runs setuid to root in order to read the audit records from the kernel.

ebsmgen — (batch and real-time modes) accepts and translates Solaris BSM audit records into EMERALD event messages as discussed in Section 4.4.1. An intermediate process is also used to manage buffers efficiently.

eXpert-BSM — (batch and real-time modes) is the EMERALD P-BEST-based forward-chaining expert system knowledge base and inference engine. It accepts event messages from *ebsmgen* and produces intrusion detection reports.

4.4.3 Alert message format

Within the EMERALD project, a format for alert messages has been developed, with both producer and consumer processes in mind. Producers include monitors targeting diverse event streams, using different analysis techniques, such as the expert system used in *eXpert-BSM* or the probabilistic model of *eBayes* [59]. Typical consumers are alert management and presentation applications, correlation engines, and components handling automated attack countermeasures.

In Figure 4.4, an example alert from *eXpert-BSM* is shown. It reports that user *bob* successfully changed file permissions on a system executable file. Messages are encoded in a host-independent binary form suited for network transportation, but printed here in text form. Response recommendations are provided here both in a verbose format targeted for presentation to human operators and in a format aimed at automated response components.

4.4.4 Multihost deployment

The host-based IDS concept is tightly coupled to the surveillance needs of individual host computing assets distributed throughout a network. This concept

```
Message ID 601 2001-02-03 01:23:19.761289 UTC:
 alert_report_ID = 3
 alert_thread_ID = 3
 alert_count = 1
 alert_gen_time = 2001-02-03 01:23:19.000000 UTC
 alert_start = 2001-02-03 01:23:19.000000 UTC
 alert_model_confidence = AL_CONFIDENCE_HIGH
 incident_class = CLASS_INTEGRITY_VIOLATION
 incident_signature = BAD_SYSTEM_BIN_MOD
 incident_description =
  Alteration to system executable
 observer_type = OB_TYPE_SIGNATURE
 observer_id = 102
 observer_stream = OB_STREAM_BSM
 observer_name = eXpert-BSM
 observer_version = 1.2
 observer_location = 192.168.2.20
 observer_src_file = realtime
 source_IParray = 192.168.1.100
 source_username = bob
 source_ruid = 0
 source_euid = 0
 source_auid = 2138
 source_pid = 6597
 target_IParray = 192.168.2.20
 outcome_generic = SUCCESS
 outcome_system_code = 0 (0x0)
 command = chmod(2)
 command_parent = /usr/bin/chmod
 resource_targetname = /usr/bin/gunzip
 resource_owner = root
 resource_owner_uid = 0
 recommendation =
   "Kill process 6597, Session ID 6542.
   Isolate and examine file [ /usr/bin/gunzip ].
   Lock out user account bob until you have
   determined who is responsible for this
   activity. Check the configuration parameter
   BSM_SYSTEM_BIN_LOCATIONS."
 recommendation_directives =
   "kill -pid 6542 -da 192.168.2.20
   lockout -uname bob -da 192.168.2.20
   fixperms -fn /usr/bin/gunzip -da 192.168.2.20
    -newattr 000
   checkcfg -da 192.168.2.20
    -name BSM_SYSTEM_BIN_LOCATIONS"
```

Figure 4.4: An example alert message from eXpert-BSM.

of distributed, lightweight sensors plays a central role in the EMERALD architecture and, consequently, the EMERALD infrastructure provides mechanisms for component configuration, message transmission, alert subscription, and distributed alert consolidation [42].

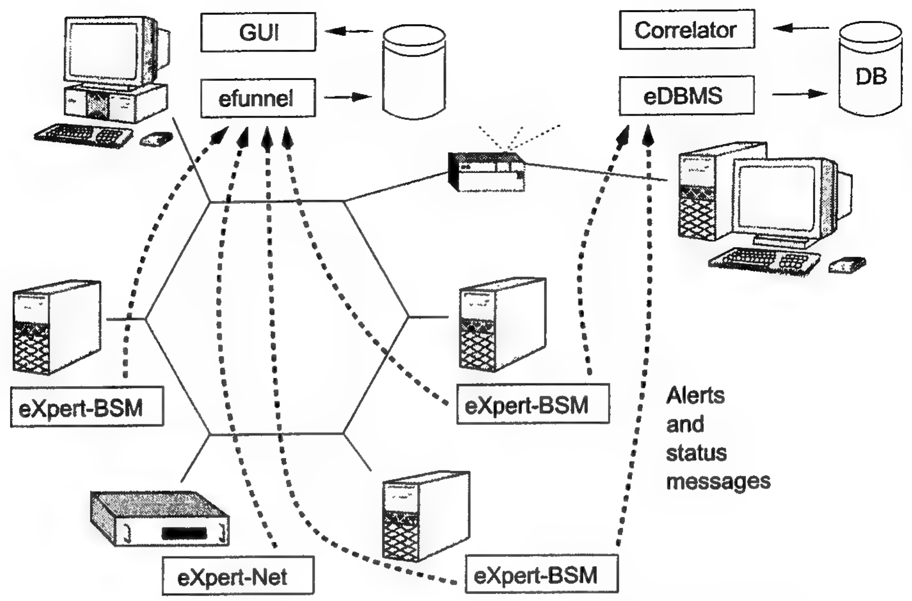


Figure 4.5: An example of multihost deployment of eXpert-BSM.

Currently, multihost deployments of *eXpert-BSM* are supported through an alert collection application called *efunnel*, which multiplexes alert and status messages from several EMERALD monitors into a single message stream. Each *eXpert-BSM* is configurable to store its produced alarms locally on its host if desired, and can simultaneously forward these alerts to other subscribing security services located on remote systems distributed throughout the network. For example, *efunnel*, or its counterpart *eDBMS* for alarm storage into an SQL database, can be deployed and configured to subscribe to alert communications from a large suite of EMERALD monitors, including multiple *eXpert-BSMs*.

An example of multihost deployment is illustrated in Figure 4.5. In this figure, EMERALD monitors are deployed across various key assets on a network. In addition to alarms, all EMERALD monitors (including *eXpert-BSM*) are capable of forwarding health and status messages to their subscribers. Health and status messages can then be used to recognize unexpected shutdown or destruction of deployed EMERALD monitors, thus making it difficult to kill a monitor without detection by other INFOSEC services. In Figure 4.5, a suite of *eXpert-BSM* monitors forward alerts to a host running *efunnel* operating on the same LAN. In addition, a subset of the *eXpert-BSM* monitors has a separate subscriber operating from a location external to the local network. The remote subscriber is managing an SQL database, and using *eDBMS* to store alert information from its selected sensors. On this remote data center, one may operate a correlation engine to examine the database of alert reports produced by selected monitors across this and other LANs.

4.4.5 EMERALD Java AlertViewer

The *eXpert-BSM* distribution provides a simple Java-based graphical user interface to manage alerts produced from its suite of distributed sensors (the alert message format is described in Section 4.4.3). The AlertViewer allows the operator to review individual alerts, manage incident handling reports, print reports, email reports, and monitor the health and status of the distributed sensors. A session history is maintained, recording actions taken on alerts by the operator. The AlertViewer consists of four windows:

Main View — displays details of alert reports produced by the EMERALD sensors, and allows operators to store administrative notes regarding each alert. This view is shown in Figure 4.6a.

Table View — displays alerts in a tabular form, where alerts can be sorted on primary and secondary attributes and quickly reviewed for alerts of greatest concern.

Status View — identifies the current suite of distributed monitors to which this AlertViewer is subscribing. Graphic indicators can identify sensors that have shut down or are late with health and status communications. As new monitors are activated, they are dynamically added to this view. This view is shown in Figure 4.6b.

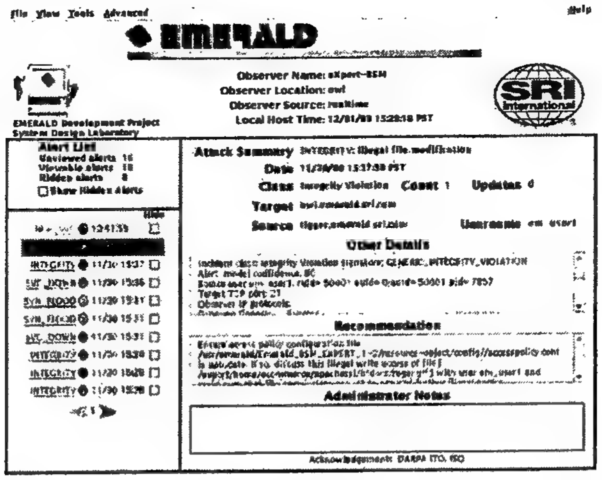
Configuration View — allows the AlertViewer operator to customize various elements of the Table view, including default sorting parameters.

4.5 Operational characteristics

eXpert-BSM has been packaged and distributed over the last 4 years to sponsors through private arrangement, and later as an evaluation release on the Internet to all who register. Throughout its development, *eXpert-BSM* has been subject to multiple third-party evaluations, batch testing, and red-team exercises as part of its participation in the DARPA Information Assurance Program. Experiences in operational deployments of *eXpert-BSM* have also greatly enhanced its capabilities, and has promoted attention to issues of optimization for real-world requirements. This section briefly summarizes activities that have enhanced the operational characteristics of *eXpert-BSM*.

4.5.1 Test battery

The *eXpert-BSM* distribution includes a test battery, containing a data collection that can be used in batch mode to trigger every intrusion model in the knowledge base. The user documentation contains both a description of the test data and a description of the expected output in terms of *eXpert-BSM* reports. In addition to being a basis for regression testing during development, this data set is useful to *eXpert-BSM* operators in several ways. Typically, an



Observer Name: eXpert-BSM
Observer Location: owl
Observer Source: realtime
Local Host Time: 12/01/00 15:20:18 PST

Alert List
Unviewed alerts 16
Viewable alerts 18
Hidden alerts 8
Show Hidden Alerts

Attack Summary INTEGRITY: illegal file modification
Date 11/30/00 15:37:38 PST
Class Integrity Violation Count 1 Updates 0
Target owl.emerald.sri.com
Source [illegible].emerald.sri.com Usernames [illegible] user1

Other Details

Incident class: integrity Violation signature: GENERIC_INTEGRITY_VIOLATION
Alert model confidence: 80
Source user [illegible] user1 ruid= 50001 euid= [illegible] gid= 50001 pid= 7857
Target TCP port 21
Observer IP protocols

Recommendation

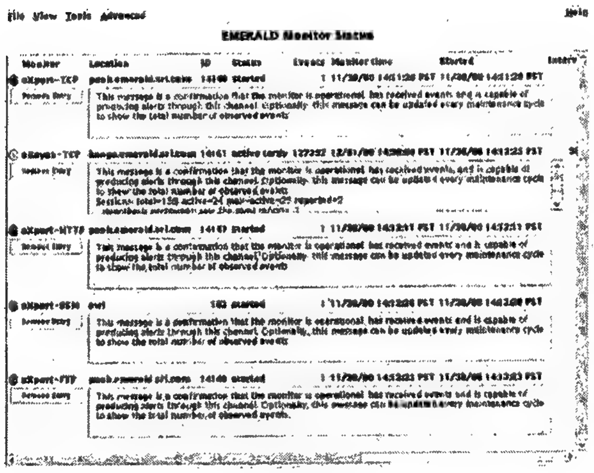
Ensure access policy configuration file
/usr/emerald/Emerald_BSM_EXPERT_1-2/resources-object/config/accesspolicy.conf
is up to date. If so, discuss this illegal write access of file [
/export/home/[illegible]] with user [illegible] user1 and

Administrator Notes

Acknowledgements: DARPA ITO, ISO

[illegible]	● 12:41:33	☐
INTEGRITY	● 11/30 15:37	☐
SU? DOWN	● 11/30 15:36	☐
SYN_FLOOD	◎ 11/30 15:31	☐
SYN_FLOOD	◎ 11/30 15:31	☐
NT_DOWN	● 11/30 15:31	☐
INTEGRITY	● 11/30 15:30	☐
INTEGRITY	● 11/30 15:28	☐
INTEGRITY	● 11/30 15:28	☐

a. Main view



EMERALD Monitor Status

Monitor	Location	ID	Status	Events	Monitor time	Started	Interv
eXpert-TCP	pooh.emerald.sri.com	14140	started	1	11/30/00 14:11:20 PST	11/30/00 14:11:20 PST	
eBayes-TCP	[illegible].emerald.sri.com	14141	active [illegible]	127232	12/01/00 14:20:[illegible] PST	11/30/00 14:13:23 PST	30
eXpert-HTTP	pooh.emerald.sri.com	14143	started	1	11/30/00 14:12:11 PST	11/30/00 14:12:11 PST	
eXpert-BSM	owl	183	started	1	11/30/00 14:12:20 PST	11/30/00 14:12:20 PST	
eXpert-FTP	pooh.emerald.sri.com	14140	started	1	11/30/00 14:12:23 PST	11/30/00 14:12:23 PST	

This message is a confirmation that the monitor is operational, has received events and is capable of producing alerts through this channel. Optionally, this message can be updated every maintenance cycle to show the total number of observed events

This message is a confirmation that the monitor is operational, has received events and is capable of producing alerts through this channel. Optionally, this message can be updated every maintenance cycle to show the total number of observed events
Sessions: total=138 active=24 max-active=25 reported=2
[illegible]

This message is a confirmation that the monitor is operational, has received events and is capable of producing alerts through this channel. Optionally, this message can be updated every maintenance cycle to show the total number of observed events

This message is a confirmation that the monitor is operational, has received events and is capable of producing alerts through this channel. Optionally, this message can be updated every maintenance cycle to show the total number of observed events

This message is a confirmation that the monitor is operational, has received events and is capable of producing alerts through this channel. Optionally, this message can be updated every maintenance cycle to show the total number of observed events.

b. Status view

Figure 4.6: Two of the views of the EMERALD Java AlertViewer.

operator would like to be assured that the monitor works correctly after installation. When started in test mode, *eXpert-BSM* loads the configuration settings that are specific for the test battery, and starts a batch mode run that will exercise the analysis and presentation components of *eXpert-BSM*. This will show the operator all the possible types of reports that the monitor can produce. In addition, if the operator wants to run live attacks to make an end-to-end test in real time, the test battery documentation provides helpful instructions.

The test battery is designed to be a concentrated collection of attack data, with as little normal data as possible, to minimize the size of the data file and the time it takes to run through it. The purpose of the test is to validate the proper operation of all the intrusion models in *eXpert-BSM*, and to demonstrate the contents of the resulting reports. Although producing such data is time-consuming and sometimes difficult, we believe that this type of test battery is appreciated by IDS operators. Results from a questionnaire sent to registered *eXpert-BSM* operators strongly support this view.

Another form of testing is to run through large amounts of normal data to make sure that there are no false alarms. We continuously perform such testing of *eXpert-BSM* in our development and production environments. The DARPA IDS evaluation data from 1998 and 1999 is of this character [34], with several weeks of large data sets containing a small suite of attacks inserted for every day. The attacks selected are intended to exploit a strategically broad range of vulnerabilities that are representative of the major threats being used to infiltrate systems today.

4.5.2 Experimentation, deployment, and evaluation

Over the years of its development, versions of *eXpert-BSM* have been deployed in third-party laboratories, such as groups within the Air Force Research Laboratory and National Security Agency, for operational evaluations and experiments. In addition, these components have participated in multiple yearly live red-team exercises, mainly within the DARPA Information Assurance & Survivability suite of research programs. These activities have provided valuable input to the continuing development of the knowledge base and other features of *eXpert-BSM.*

In April 2000, the first release of *eXpert-BSM* was made available for download on the Internet. Those who registered their contact information were granted a time-limited evaluation license. More than 200 organizations have registered.

We are currently aware of at least one military operational center and one commercial data center where the evaluation version has been fielded operationally to monitor critical servers. At both centers *eXpert-BSM* has been in continuous use for more than a year.

4.5.3 Optimization and performance

In general, expert systems built with P-BEST are much faster than systems using the traditional interpreting model, because P-BEST code is translated to C, which is then compiled just like any C program [32]. In addition, P-BEST has undergone several modifications to further enhance its performance in terms of speed and integration with other programs. The modifications include language extensions that allow most C native types to be used in P-BEST, translator directives to pass some constructs directly to the C code, and an improved execution model for the inference engine. We have also developed C libraries that optimize the evaluation of complex antecedent expressions.

For any IDS analyzing a high-bandwidth event stream, it is important to be able to discard as much irrelevant data as possible as early in the process as possible. The *eXpert-BSM* knowledge base uses only 58 of the more than 250 possible types of BSM audit records (auditable event types) in its intrusion models. In real-time mode, the Solaris audit kernel module is configured to produce only those 58 types of records. For batch mode, our preprocessing component *ebsmgen* performs the same selection. Our experiments show that for large sets of typical audit data (> 1 GB), this preselection reduces the amount of data that needs to be produced and processed to on average about 10% of the total amount that would be produced if full auditing were enabled.

The original auditd is designed to write audit records only to files. The *eXpert-BSM* package includes a component called *ebsmprobe* that replaces auditd, reads audit records directly from the kernel, and uses interprocess communication to pass the records to the preprocessing and analysis components for direct consumption. Thus, *eXpert-BSM* avoids expensive disk I/O operations for audit records and eliminates the need to reserve large amounts of disk space for audit files.

We recommend installing *eXpert-BSM* on local disk space rather than on network-mounted volumes, for better security and to avoid unpredictable file access delays. Internally, any kind of over-the-network access such as NIS or DNS lookup is avoided, except during the short initialization phase. Because many sites use NIS for user account information, *eXpert-BSM* uses its own local file for mapping numerical user IDs to usernames, which comprises the information in */etc/passwd* and NIS.

If the monitored host is running an extremely active process producing very large volumes of audit records, such as a heavily loaded DBMS, auditing can be turned off for that process to let the IDS be more responsive in its monitoring of the other processes on the host. We propose that a separate account be created for the sole purpose of running the heavy process, and that the account be excepted from auditing by an entry in */etc/security/audit_user*.

To obtain performance measurements, we have deployed *eXpert-BSM* on a Sun Enterprise 450, which is used as a file server and compute server for about 15 users. The machine is equipped with two UltraSparcII 400 Mhz processors, and 1 GB RAM. The additional load imposed by *eXpert-BSM* was studied in an experiment where we measured the completion time for building a relatively large software package, both in the presence and in the absence of the *eXpert-BSM* monitor. We ran *make* for a clean distribution of openssl-0.9.6 and measured the completion time as reported by */usr/bin/time*. A total of 10 runs were performed for each of the two situations, and each run was followed by other operations to eliminate the effects of file-system caches and so forth. When *eXpert-BSM* was not running, the 10 builds took on average 428 seconds each to complete, with a standard deviation of 0.8. With *eXpert-BSM* running in its "out-of-the-box" configuration, each build produced 94,684 audit event records, and took on average 454 seconds to complete, with a standard deviation of 1.1. We can conclude that the presence of the *eXpert-BSM* monitor caused a 6% increase in completion time for the task.

4.6 Related work

Operating system audit logs offer an interesting vantage point to the security-relevant operations of host systems. In [44], a design of effective auditing for security-critical systems is explored. Some standardization efforts for handling audit content have been examined [7], as have issues of what additional network-related activity is worthy of representation in host audit trails [11]. A more recent work on applying formality to audit log structures is [18], which includes a discussion on some of the difficulties in automated BSM audit trail parsing.

Various related research efforts explore what one can do with audit data to automatically detect threats to the host. An important work is MIDAS [53], as it was one of the original applications of expert systems—in fact using P-BEST—to the problem of monitoring user activity logs for misuse and anomalous user activity. CMDS, by SAIC, demonstrated another application of a forward-chaining expert-system, CLIPS, to a variety of operating system logs [47]. USTAT [23] offered another formulation of intrusion heuristics using state transition diagrams [45], but by design remained a classic forward-chaining expert system inference engine. ASAX [21] introduced the Rule-based Sequence Evaluation Language (RUSSEL) [39], which is tuned specifically for the analysis of host audit trails.

4.7 Conclusions

Host-based intrusion detection offers the ability to detect a wide variety of computer misuse through the direct analysis of process activity inside the host. Host-based analysis offers an important complement to network traffic analysis, providing threat detection coverage that is simply not easily available through the analysis of raw network traffic.

eXpert-BSM is a powerful and mature service for isolating security misuse and important security-relevant warning indicators. It analyzes the rich content of the Solaris BSM audit stream in real time, providing operators with distilled alert information and response recommendations. *eXpert-BSM* has been under development since 1998, and continues to progress in its effectiveness and usability through extensive testing, experimentation, and deployment experience.

eXpert-BSM is available for download at:

`http://www.sdl.sri.com/emerald`

Chapter 5

Application-based Detection

This chapter describes how data collected inside critical applications can be used for intrusion detection. In addition to the present project, the work on the Web server module [2] was supported by DARPA/AFRL under contract number F30602-99-C-1049.

5.1 Introduction

Intrusion detection systems (IDSs) can be categorized with respect to several different dimensions, of which the commonly used (but somewhat oversimplified) dichotomy between *misuse detection* and *anomaly detection* is one example. Another dimension for categorization is the type of event data analyzed by the IDS. An IDS that monitors traffic flowing in a network is usually called *network based*, while an IDS that analyzes data produced locally at a host is often referred to as *host based*.

We subdivide the host-based category further, depending on the abstraction level at which data is collected. Most existing host-based systems gather audit data at the operating system (OS) system-call level, but an IDS could get its data from higher as well as lower abstraction levels. Below the OS level, we could, for example, look at the executed processor instructions. Above the OS level, we could collect data from service applications such as database management systems, Web servers or e-mail daemons, or from end-user applications. As different types of security violations manifest themselves on different levels in a system, one could argue that it is important for the IDS to collect data at the most meaningful abstraction level(s) for the event in question. It should be kept in mind that independent of the type of data collected, it can be sent to any type of analysis engine (e.g., signature based, model based, probabilistic).

In this chapter, we focus on collection of data produced by applications (above the OS level) and refer to an IDS analyzing such data as *application* *based.* Although the concept of application-based IDS is not new, there is a striking absence of commercial IDSs for applications other than firewalls [25]. The approach presented in this chapter shows how the data collection for an application-based IDS can be integrated with the monitored application.

The remainder of this chapter is organized as follows. Section 5.2 discusses limitations of network-based and host-based IDSs, respectively. In Section 5.3, we present our application-integrated approach, and discuss its advantages and how it complements the other methods. Section 5.4 describes an implementation for a Web server to validate our reasoning. In Section 5.5, we examine the performance characteristics of the implementation. Section 5.6 describes related work, while ideas for future work are outlined in Section 5.7. Our conclusions are summarized in Section 5.8.

5.2 Background

Many researchers have recognized that there is no single "silver bullet" approach to automatic detection of security violations. By combining and integrating complementary approaches, better attack space coverage and accuracy can be achieved. In this section, we look at some specific problems with the network-based and host-based approaches, respectively.

5.2.1 Network-based data collection

The popularity of Ethernet technology paved the way for network-based IDSs. When traditional broadcast Ethernet is used, a whole cluster of computers can be monitored from a dedicated machine that listens to all the traffic. As no changes in the infrastructure are required, and there is no performance penalty, most free and commercial IDSs use this approach. The system can be completely hidden by having a network card that listens to the network but never transmits any traffic itself. However, by decoupling the system from the actual hosts it supervises, we lose valuable information.

First, probably the most serious problem with the network-based approach is encrypted traffic, which in effect makes the network monitor blind to many attacks. Today, encryption is typically used to protect the most sensitive data, and there are indications that encryption will become more ubiquitous in the near future, making today's network-based monitors ineffective.[1]

Second, the IDS can be deceived in several ways. Most Internet standards in the form of RFCs carefully define valid protocol syntax, but do not describe in detail how the application should behave with deviant data. To be accurate, the IDS needs to model how the application interprets the operations, but

[1]There are some ways a network-based IDS could read encrypted traffic. For example, it could act as a proxy with the encrypted channel being only to the IDS (or a similar proxy), thus introducing unnecessary overhead in the form of extra programs that need to be supervised and also exposure of data before it reaches its final destination. The network monitor can also be given the private key of the server, increasing the exposure of the key and forcing the network monitor to be able to keep track of user sessions and their associated keys.

this is almost an impossible task without receiving feedback from the application. Minor differences in operations play a major role in how they are interpreted. For example, consider a user requesting a certain Web page, by sending `http://www.someHost.com/dir1\file1` (note the backslash character). If the receiving Web server is Microsoft-IIS/5.0 under Microsoft Windows, the server looks for the file `file1` in the directory `dir1`. However, if the server is Apache version 1.3.6 under Unix, the server looks for a file named `dir1\file1` in the root directory. Even if an IDS could model the different popular servers, it cannot account for every implementation difference in every version of these. This problem is not limited to application-level protocols, but as Ptacek and Newsham [48] describe, the same goes for lower-level protocols.

Third, efforts to increase bandwidth in networks pose serious problems for network-based IDSs. Higher line speed is in itself a difficulty, and switching (unicast) technology ruins the possibility to monitor multiple hosts from a single listening point. Some IDS developers try to address this problem by placing a network data collection unit on every host. That solves this problem while introducing others, which are similar to the problems faced by host-based analysis.

5.2.2 Host-based data collection

The host-based approach addresses some of the problems described above, with the primary advantage being access to other types of information. As it is installed on a host, it can monitor system resources as well as look at operating system audit trails or application logs. It is also independent of the network speed as it monitors only a single host. However, the system administrator now needs to install a number of monitors instead of just one, thus incurring more administrative overhead. Also, the user could experience a performance penalty as the monitor is on the same host as the application.

Furthermore, most monitors on the OS level cannot detect attacks directed at the lower network protocol levels because network information typically does not become available in the audit event stream until it has reached the higher protocol levels. See [11, 12] for an approach to include network data in OS audit data.

An application-based monitor in the traditional sense (such as the one described in [1]) reads data from log files or other similar sources. By the time the information is written to the log, the application has completed the operation in question and thus this monitor cannot be preemptive. The information available is often also limited to a summary of the last transaction. For example, a Web request in the Common Logfile Format (CLF) is

```
10.0.1.2 - - [02/Jun/1999:13:41:37 -0700] "GET /a.html  HTTP/1.0" 404 194
```

Without going into the meaning of the different fields, the following recounts the scenario: The host with address 10.0.1.2 asked for the document `a.html`, which at that time did not exist. The server sent back a response containing 194 bytes.

The log entry does not contain all the information an IDS needs for its analysis. Were the headers too long or otherwise malformed? How long did it take to process the request? How did the server parse the request? What local file did the request get translated into?

In some applications, logging can be customized and contain much more information. Nevertheless, we have not yet seen a system where all internal information needed to understand the interpretation of an operation is available for logging. Furthermore, by turning on all log facilities, we increase the risk of running out of storage space for the logs and incurring performance degradation.

5.3 Application-integrated data collection

As we have shown in the previous section, there are problems associated with both network-based and host-based approaches. Some of these can be solved by collecting data directly from the single critical application that we want to monitor. In this section, we present the general principles of this approach, while Section 5.4 describes a prototype implementation for monitoring Web servers.

5.3.1 Rationale

Today's network structure within organizations makes a few applications critical. These need to be available around the clock from the outside of the organization; they are sensitive to attacks but seldom sufficiently protected. Examples include Web servers, e-mail servers, FTP servers, and database systems.

To minimize security concerns, most such critical applications run on dedicated machines and are protected by firewalls. Typically, no other application is running on these machines. If remote login is allowed, it is very restricted (such as only ssh). Thus, the malicious user must go through the channels of the critical application to subvert the host. By having the IDS monitor the inner workings of the application, analyzing the data at the same time as the application interprets it, we have a chance of detecting malicious operations and perhaps even stopping them before their execution. However, for us to successfully integrate a monitor into the application, the application must provide an interface. Some applications provide an API and, as the advantage with an application-integrated monitor becomes clear, we hope that more vendors will provide such interfaces. Other venues for integration are found in the open-source movement.

5.3.2 Advantages

Access to unencrypted information

In almost all cases, data must be accessible inside the application in unencrypted form for meaningful processing, even if it is encrypted in lower layers. Consequently, the unencrypted data is also accessible to an application-integrated data collection module. This is a major advantage compared to a network-based IDS. Moreover, it should be noted that because encryption is used for the most sensitive data, the functions handling that data are probably among the most interesting from an attacker's point of view and therefore important to monitor.

Network speed is not an issue

The module is part of the application, and takes part in the normal processing cycle when analyzing operations. Thus, the limiting factor is the application speed rather than the network speed. For example, if the original application can accept a certain number of connections per second, the application equipped with the module must be able to perform equally well. Care must be taken so that the module does not become a bottleneck and does not consume too many of the host's resources. We discuss our solution to this problem in detail in the next section.

More information available

Being part of the application, the module can access local information that is never written to a log file, including intermediate results when interpreting operations. It can monitor how long it takes to execute a specific operation, to detect possible denial-of-service attacks. Furthermore, we expect an application-integrated monitor to generate fewer false alarms, as it does not have to guess the interpretation and outcomes of malicious operations. For example, a module in a Web server can see the entire HTTP request, including headers. It knows which file within the local file system the request was mapped to, and even without parsing the configuration file of the Web server, it can determine if this file will be handled as a CGI program (not otherwise visible in either network traffic or log files).

True session reconstruction

Information of interest to an IDS often concerns transactions (request–response) and user sessions. To extract that information, a network-based monitor must perform expensive reconstruction of transactions and sessions from single network packets, and it is still not guaranteed to correctly mimic the end-point reconstruction. In contrast, the application-integrated module is handed complete transaction and session records directly from the application, and there is consequently no discrepancy between the interpretations.

The IDS could be preemptive

IDSs are at times criticized for being of limited use as they can only report intrusions, usually too late to stop any damage. By being part of the application, the module could supervise all steps of the processing cycle and could react at any time. For example, it could deny a single operation that appears malicious without otherwise compromising server performance.

5.3.3 Disadvantages

The disadvantages of the application-integrated monitor coincide with some of the disadvantages of the host-based monitors in general, as described in Section 5.2.

Any monitoring process running on the same host as the monitored service risks impacting the performance of the server. It is therefore important that a goal in the monitor design and implementation is to minimize this performance impact.

Given that one needs to have a distinct application-integrated monitor for every single type of application one wants to monitor, the development efforts could be significant. However, in today's situation where a handful of products dominate the field of network server applications, the efforts and costs required for satisfactory coverage could be lower than they might first appear. This is particularly true for applications that are open source and/or provide an API for modules.

The application-integrated monitor can only be a complement to other types of IDSs. As it is part of the application, it sees only the data reaching the application. By targeting a protocol below the application layer, an attacker could evade detection by our module, but would be within the scope of a network-based IDS or possibly another host-based sensor specialized in lower-level protocols.

5.4 Design principles and implementation

As discussed in the previous section, current network infrastructure makes a few applications critical. Of these, the Web server stands out as being both ubiquitous and vulnerable. First, most organizations need a Web server, and it is the service most users associate with the Internet. Second, even though the server software might be as stable (or unstable) as other types of software, many sites have customized programs connected to the server allowing access to legacy database systems. Because these programs are easy to write, they are often developed by junior programmers who do not properly consider the security aspects of letting any user in the world supply any input to programs run locally. As a result, new vulnerabilities in CGI programs are discovered daily. Furthermore, the Web server is among the first to be probed during a reconnaissance. The vulnerabilities are easy to comprehend (e.g., add a semicolon after the request in your browser), and the gratification is instant (change the Web pages and all your friends can admire your deed).

For these reasons, we chose to focus our prototype on a Web server. The major Web server brands also have an API that provides the capabilities we

Table 5.1: Market shares for the top Web server products [41]

Product	Developer	Market Share
Apache	Apache	60.0%
Microsoft-IIS	Microsoft	19.6%
Netscape-Enterprise	iPlanet	6.2%

desire for application-integrated event data collection. The remaining question is which server product to target.

Netcraft continuously conducts a survey of Web servers on the Internet [41]. Some results from the February 2001 survey are shown in Table 5.1. It shows that there are three main players in the market covering more than 85%. We decided to build our first application-integrated data collection module for the Apache Web server, as it is the most popular one.

The market penetration for SSL is very different. Unfortunately, the data is considered commercially valuable and is available as a commodity only for a prohibitively high price. Furthermore, the Netcraft survey counts each site equally. This reflects neither the popularity of the site nor the risk and associated cost of attacks. We are not aware of any other survey of this kind, and even if the numbers above are subject to discussion, there is no doubt that Apache has a large share of the market.

5.4.1 Implementation

As it turns out, it is quite easy to extend the Apache server with our data collection module, primarily due to the following reasons.

- There is a well-defined API for modules, and the data concerning each request is clearly distinguishable in distinctive data structures.
- Each request is processed in several stages, and there are so-called hooks or callbacks in each of these stages that a module can use (see Figure 5.1).
- After each stage, the module can give feedback to the server as to whether it should continue executing the request.
- There is support for extensive logging capabilities through the reliable piped logs interface (explained below).

We built the module within the framework of EMERALD [46], which among other components include the eXpert-HTTP analysis engine and the eFunnel communications process described below. The layout of our system is depicted in Figure 5.2. For each request, the following happens:

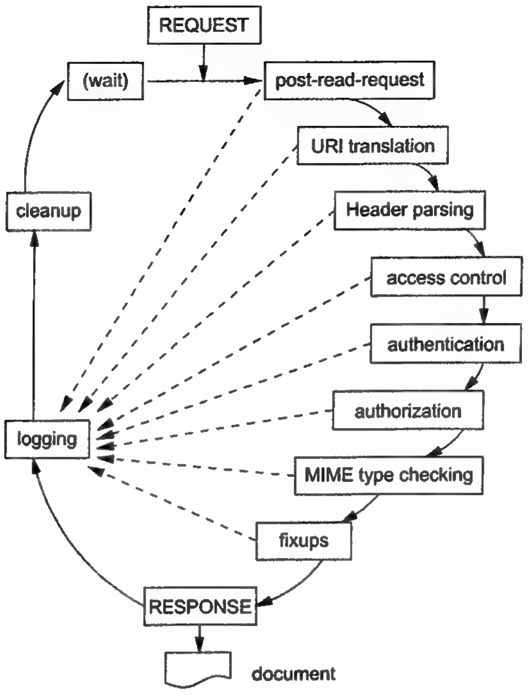


Figure 5.1: The Apache request loop [54]. The solid lines show the main transaction path, while the dashed lines show the paths that are followed when an error is returned

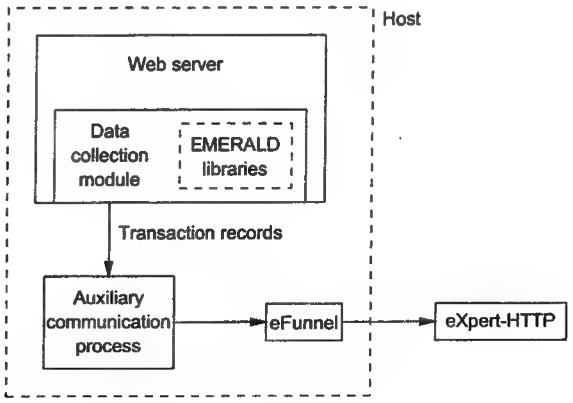


Figure 5.2: The architecture and data flow in the implementation

1. The Web server passes control to our module in the logging stage of the request cycle.

2. The module takes relevant data of the request and packs it into a format the analysis engine eXpert-HTTP can understand.

3. Through the reliable piped logs interface of Apache, we pass the information to an auxiliary program, which just hands the information to a second program, eFunnel, through a socket.

4. eFunnel, in turn, communicates with an eXpert-HTTP on an external host.

5. The eXpert-HTTP performs the analysis.

Below we discuss each of these steps in detail. The design reflects mainly three concerns. Most important, we do not want to introduce vulnerabilities into the server software. For this reason, we decided to keep as little code as possible within the server. The second issue is performance. If the module makes the server slow, it will not be used. By limiting the analysis on the server host, we gain speed but we lose interactivity between the module and the server. Third, we wanted to reuse as much code as possible, both in the server and from the EMERALD system.

As our major concern was the risk of introducing vulnerabilities into the server, we decided against letting the analysis engine be part of the server. This would have added a lot of extra code, thus increasing the complexity and making the server more vulnerable to bugs [10, Theorem 1]. Although the analysis engine could have been placed on the same host, we wanted to demonstrate that larger sites can use this approach and satisfy critical performance requirements. However, removing the analysis engine from the server in turn means we limit the preemptive capabilities we described in Section 5.3, as the distance introduces latency between the receiving of a request and the final analysis. In Section 5.7, we propose a two-tier architecture that offers an acceptable tradeoff between the ability to react in real time while still minimizing performance penalties. Note that the setup with the server existing on a separate host is more complicated than having the analysis engine on the same host. On sites where performance is not of critical importance, we recommend the simpler approach.

The introduced latency described in the previous paragraph restricts the reactive capabilities of the module. For this reason, we decided to let our module be called only in the last step of the request cycle—the logging step. By this time, the server has interpreted the request and sent the data to the client, and all information about the request is available for logging, which makes it easy for our module to extract it. Even though this seems to be marginally better than a system reading log files, we would like to point out two advantages with our proposed application-integrated module.

First, we have access to more information. For example, consider the request *http://www.w3c.org/phf.* The application-integrated monitor can determine if the server will handle the request as a CGI script (possibly bad), or if it accesses an HTML file (that, for example, describes the phf attack). Second, the information is immediately available with the application-integrated approach. A monitor watching a log file must wait for the application to write the information to the file, the caching within the operating system, and possibly the next monitor polling time.

The last steps of our design are explained by considering code reuse. Within the EMERALD framework, we have a component called eFunnel. This program accepts incoming connections where EMERALD messages can be transmitted, and passes the information to outgoing connections. It can duplicate incoming information (e.g., having two different analysis engines for the same application) or multiplex several incoming flows into one outgoing connection (e.g., comparing the results of a network-based monitor with an application-integrated monitor for discrepancies). This program takes into account problems that might appear in interprocess communication, such as lost connections or necessary buffering. Thus, eFunnel exactly matches our needs to send information from the module to other components.

On the server side, Apache provides a reliable pipe log interface. This interface sends the log information directly to a program. The term *reliable* signifies that Apache does some checking on the opened channel, such as making sure the connection is still accepting data. If that is not the case, it restarts the program [54, p. 563]. We also hope to capitalize on future advances within the implementation of this interface.

As noted, we would like to use both eFunnel and Apache's "reliable log format" but we run into a practical problem. In our tests, Apache started the receiving program twice [54, p. 59], but eFunnel binds to certain predefined sockets locally, and can thus be started only once. Our solution involves the auxiliary program described in Step 3 above, which provides a clear interface between the Web server and the IDS. Apache can restart this program as often as necessary, without the IDS being affected.

5.4.2 Analysis engine

Within EMERALD, we are developing a package of network-based IDS components, one of them being *eXpert-HTTP*, an analysis component for HTTP traffic. It was developed to receive event messages from the network data collection component *etcpgen*, but can equally well receive the messages from the application-integrated module. Actually, we can use exactly the same knowledge base independent of the source of the data and no additional development cost is necessary for the analysis engine. After we had completed the data collection module, we could directly test it by having it send the data to eXpert-HTTP. It is not surprising that as more information is available through the module than through network sniffing, we have the possibility of constructing new rules to detect more attacks as well as refining existing rules to produce more accurate results.

To summarize, our module extracts all transaction information from the Web server and packs it into a format that the analysis engine can understand. The module then ships off the information (through multiple steps due to the aforementioned implementation issues) to the analysis engine located at a separate host. No changes to the analysis engine were necessary even for detection of attacks using the encrypted SSL channel, so the module and the network-based event data collector could be used interchangeably with the same knowledge base. If we want to capitalize on the extra information we gain by using the module, we obviously need to develop new detection rules.

5.5 Monitor performance

From a performance standpoint, the application-integrated module does not do anything computationally intensive. It simply accesses a few variables and formats the information before it is sent on. This should not decrease the server performance. However, we would like to have a more substantial claim that our approach is viable. One way would be to let the module include a call to a timing function (in C) to show how much execution time is spent inside the module. We decided against this measure, as we are more interested in measuring the user experience when an entire monitor is running. This means that we configured the module to send the data to the eXpert-HTTP analysis engine on another host, as depicted in Figure 5.2.

WebLoad from RadView Software is an advanced Web application testing and analysis package [49]. Among its many features and options, it allows us to specify a single URL that will be continuously fetched during a specified time interval. WebLoad measures the round-trip time for each transaction, giving a fair measure of the user experience with the network and the server. We used the free evaluation version of WebLoad, which has some restrictions compared to the full-blown commercial version. However, those restrictions (no support for SSL and a maximum of 12 virtual clients) did not significantly limit our ability to evaluate the performance of our module.

We set up four runs, each lasting 60 minutes and using 10 virtual clients on a single physical client host. We used two different types of URLs, the first returning a Web page consisting of about 50 KB of text and a 12 KB JPEG image, while the second caused the execution of a CGI program. A summary of the results is in Table 5.2. The absolute values may seem somewhat high, but are due to the relatively low-end server hardware configuration. The relative difference between running the monitor or not is so small that it is probably only caused by the CPU load imposed by the auxiliary communications process and eFunnel running in parallel with the Web server. In fact, we believe that the results confirm that a future application-integrated module can have *zero* performance impact (with respect to response time) on the Apache application, as explained below.

In Figure 5.1, we show the request cycle for the server. The module performs logging *after* the request has been sent back to the client. For this reason, we expect that it is possible to postpone the penalty of the module until the next

Table 5.2: Performance measurements

Round-trip time (seconds)	Page without monitor	Page with monitor	Impact	CGI without monitor	CGI with monitor	Impact
median	1.486	1.517	2.1%	1.192	1.229	3.1%
average	1.499	1.521	1.5%	1.195	1.238	3.6%
std. deviation	0.057	0.059		0.034	0.048	

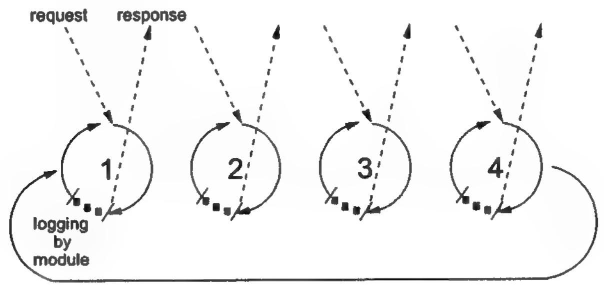


Figure 5.3: Details of a possible scenario for Web server request handling. The numbers denote parallel processes ready to serve a new request

request. In Figure 5.3, we show details of a possible scenario of what could happen in the server. The server spawns several child processes to handle new requests. As long as there are enough children, the penalty of the module does not need to be noticeable, as a child can finish logging before it receives the next request to handle. Under heavy load, where a request would have to wait for logging of the previous request, this would of course be different.

Apache/1.3.6 behaves in a different way from what is depicted in Figure 5.3. Depending on the content sent back to the client, we observed a different behavior. We suspect that this depends on different implementations of the so-called Content-Handlers [54].

We did not stress test the Web server in the measurements described above, for two reasons. First, we have measured the whole system and we cannot attribute the difference in time to any specific reason. A stress test also affects the server and the operating system, neither of which is under investigation in this report. Second, we believe that many commercial server parks are built for peak needs and are therefore normally underutilized.

5.6 Related work

The concept of application-based IDSs is not new, and there are indications that this would be the next big field to explore for commercial IDS vendors. There are some IDSs that are capable of monitoring firewalls [25]. In contrast, we are currently aware of only a single example of a commercial Web server IDS, AppShield from Sanctum, Inc. [51]. It is a proxy that overlooks the HTML pages sent out. For example, it scans HTML forms sent from the server, and makes sure that the information returned is valid (e.g., allows the use of hidden fields). Even though it actively monitors the Web server, it is not integrated into a greater IDS framework.

The closest to our approach is the work done by Burak Dayioglu [13]. His module simply matches the current request with a list of known vulnerable CGI programs. As the analysis is performed in line with the data collection, there is a greater risk of reduced server performance, especially if the list grows large.

5.7 Improvements

The module prototype described in this chapter is the first step in the exploration of the possibilities of application-integrated data collection for IDS. By extending the knowledge base of the IDS to fully utilize the information available through the module, we hope to improve detection rates and reduce false-alarm rates. A natural step is also to develop similar data collection modules for other server applications, such as FTP, e-mail, databases, and Web servers other than Apache. We already have a working prototype for the iPlanet Web server.

It could also be interesting to have an analysis engine compare the transaction data from the application-integrated module with data from network sniffing. The subset of data items that is available to both collectors should normally be identical, except when someone actively tries to fool one of the collectors. This could potentially enable detection of advanced attacks that try to circumvent the IDS.

In Section 5.4, we described the trade-off between preemptive capabilities (application-integrated analysis) and low performance impact (application-integrated data collection but external analysis). We could have a two-tier architecture where the application-integrated module performs a quick and simple analysis before the request is granted by the application and then passes the data on to an external advanced analysis engine. If the first analysis detects an attack attempt, it could cause the request to be denied. The second analysis could also pass feedback to the first, such as the name of a host that has launched an attack and should not be serviced again. Instead of outright denying the request, the application-integrated module could suspend a request it finds suspicious until the external analysis engine has reached a conclusion. By tuning the first filter, we could restrict the performance penalty to a small subset of all requests, but still be able to thwart attacks.

5.8 Conclusions

We have presented an application-integrated approach to data collection for intrusion detection. By being very specialized, the module is closely tailored to be part of the application and have access to more information than external monitors, thus being able to discover more attacks but also to reduce the number of false alarms. Because each module is specific to one application product, coverage of many products could lead to an increased development cost, but we showed several reasons why that is not a severe limitation of this approach.

If this approach becomes popular, we expect vendors to provide an API for similar products in the future to stay competitive. This means very little extra effort is needed to include this type of monitor in a component-based IDS, such as EMERALD. We also showed the advantages with our prototype for the Apache Web server. It gave us access to information internal to the server, which helped the IDS understand how the server actually parsed the request. The module had access to the decrypted request, even if it was transported through SSL over the wire. As we clearly separated the data collection from the analysis engine, the performance penalty was negligible. The knowledge base could be used with no change, thus leveraging previous investments.

Chapter 6

Evaluation and Experimentation Summary

This chapter briefly summarizes our participation with EMERALD monitors in various evaluations and experiments, as part of the DARPA program.

6.1 The DARPA evaluations 1998 and 1999

As part of the DARPA Information Assurance and Survivability program, MIT Lincoln Laboratories was given the task to evaluate IDS efforts within the program. This resulted in the so-called off-line evaulations of 1998 [35] and 1999 [34]. The EMERALD program participated successfully in both those evaluations. In the 1998 evaluation, an emphasis was placed on SRI's early prototype involving statistical anomaly detection (eStat). The main lesson learned from the 1998 evaluation was that significantly increased attack coverage was needed and, as a result, we moved toward a more extensive development of our eXpert-based technology. That led to our success in the 1999 evaluation, where the EMERALD monitors outperformed the other participants in most categories.

The large data sets produced for these two evaluations has been most useful as reference material in the continuous development of the EMERALD monitors.

6.2 Integration experiments

The EMERALD monitors have participated successfully in several experiments at the Technology Integration Center (TIC) where our monitors have been integrated in network defense structures that were exposed to red teams [29]. These exercises have provided useful insight not only to the development of detection capabilities, but also to the design of user interfaces and other operation-related features.

6.3 Operational evaluation

In May 2000, the eXpert-NET and eXpert-BSM monitors participated in an operational evaluation at a large military installation in the Pacific [5]. eXpert-Net performed well compared to other research and commercial systems, while eXpert-BSM performed exceptionally well with a detection rate of 100% and zero false positives in the experiment. In fact, the operators decided to keep eXpert-BSM in operation at their site. This experiment shows that eXpert-BSM is ready for operation deployment and that it covers a previously dark space in intrusion detection, namely real-time audit-based analysis.

Chapter 7

Concluding Remarks

We have shown how the EMERALD monitors are able to detect various forms of misuse by analysis of a diverse set of data streams on different abstraction levels, and to communicate their findings in a common alarm management infrastructure.

Overall, the progress to date in developing and using EMERALD has been very promising. The development and integration of intrusion detection engines for a variety of event streams has laid the groundwork for interesting research on new fields related to alert correlation, attack scenario recognition, intrusion tolerance, and automated response. In particular, we note the following:

- The software engineering practice used in EMERALD's modular design and the attention devoted to well-defined interfaces and information hiding in the sense of David Parnas have proven very valuable in EMERALD's development thus far, and will be even more valuable to the ability to interoperate with components developed elsewhere, to its long-term evolvability, and to subsequent generalizations of EMERALD beyond security applications to address human safety, enterprise survivability, reliability, real-time performance, and other critical attributes.

- Hierarchical and distributed correlation is necessary in analyzing highly distributed environments, because of the inability to recognize global patterns from isolated local events. However, additional analysis techniques are likely to be required.

- The iterative nature of EMERALD instantiations will enable lightweight detection components to specialize in particular areas of concern, for different event spaces and at different layers of abstraction.

A few general conclusions are also noted in an attempt to put the EMERALD experience in perspective.

- Commercial intrusion detection systems have concentrated mostly on string matching and other forms of simple signature identification to detect classes of outsider attacks, often with very high false-alarm rates. To date, primarily the easy parts of the problem have been addressed by the commercial community.

- Detecting, identifying, and responding to hitherto unknown attacks and anomalies remain as very challenging problems, including highly coordinated attacks, subtle forms of misuse by insiders, and anomalous network behavior resulting from malfunctions and outages. Providing global rather than local analysis is still a very important research area that is relatively uncharted. Generalizations beyond known security attacks are also challenging.

Bibliography

[1] M. Almgren, H. Debar, and M. Dacier. A lightweight tool for detecting web server attacks. In *Proceedings of the 2000 ISOC Symposium on Network and Distributed Systems Security*, pages 157–170, San Diego, California, Feb. 2–4, 2000.

[2] M. Almgren and U. Lindqvist. Application-integrated data collection for security monitoring. In W. Lee, L. Mé, and A. Wespi, editors, *Recent Advances in Intrusion Detection (RAID 2001)*, volume 2212 of *LNCS*, pages 22–36, Davis, California, Oct. 10–12, 2001. Springer-Verlag.

[3] D. Anderson, T. Frivold, and A. Valdes. Next-generation intrusion-detection expert system (NIDES). Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, Menlo Park, California, May 1995.

[4] D. Anderson, T. Lunt, H. Javitz, A. Tamaru, and A. Valdes. Safeguard final report: Detecting unusual program behavior using the NIDES statistical component. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, Dec. 2, 1993.

[5] S. Arnold. Advanced intrusion detection experiment final report. Technical Report D658-10944-1, Boeing Defense & Space Group, PO Box 3999, M/S 88-12, Seattle, WA 98124-2499, Sept. 13, 2000.

[6] S. Axelsson, U. Lindqvist, U. Gustafson, and E. Jonsson. An approach to UNIX security logging. In *Proceedings of the 21st National Information Systems Security Conference*, pages 62–75, Arlington, Virginia, Oct. 5–8, 1998. National Institute of Standards and Technology/National Computer Security Center.

[7] M. Bishop. A standard audit trail format. In *Proceedings of the 18th National Information Systems Security Conference*, pages 136–145. National Institute of Standards and Technology/National Computer Security Center, Oct. 10–13, 1995.

[8] D. Bruschi, E. Rosti, and R. Banfi. A tool for pro-active defense against the buffer overrun attack. In J.-J. Quisquater et al., editors, *Computer*

Security – Proceedings of ESORICS 98, volume 1485 of *LNCS*, pages 17–31, Louvain-la-Neuve, Belgium, Sept. 16–18, 1998. Springer-Verlag.

[9] CERT Coordination Center, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890, USA. *Buffer Overflow in Sun Solstice AdminSuite Daemon sadmind*, Dec. 14, 1999. CERT Advisory CA-1999-16, *http://www.cert.org/advisories/CA-1999-16.html.*

[10] W. R. Cheswick and S. M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker.* Addison-Wesley, 1994.

[11] T. E. Daniels and E. H. Spafford. Identification of host audit data to detect attacks on low-level IP vulnerabilities. *Journal of Computer Security*, 7(1):3–35, 1999.

[12] T. E. Daniels and E. H. Spafford. A network audit system for host-based intrusion detection (NASHID) in Linux. In *Proceedings of the 16th Annual Computer Security Applications Conference*, New Orleans, Louisiana, Dec. 11–15, 2000.

[13] B. Dayioglu, Mar. 2001. *http://yunus.hacettepe.edu.tr/~burak/mod_id/.*

[14] J. de Haas. Vulnerability in Solaris ufsrestore. Bugtraq, June 14, 2000. *http://archives.neohapsis.com/archives/bugtraq/2000-06/0114.html.*

[15] H. Debar, M. Becker, and D. Siboni. A neural network component for an intrusion detection system. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 240–250, Oakland, California, May 4–6, 1992.

[16] D. E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, Feb. 1987.

[17] D. E. Denning and P. G. Neumann. Requirements and model for IDES—a real-time intrusion detection expert system. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA 94025-3493, USA, 1985.

[18] C. Flack and M. J. Atallah. Better logging through formality: Applying formal specification techniques to improve audit logs and log consumers. In H. Debar, L. Mé, and S. F. Wu, editors, *Recent Advances in Intrusion Detection (RAID 2000)*, volume 1907 of *LNCS*, pages 1–16, Toulouse, France, Oct. 2–4, 2000. Springer-Verlag.

[19] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, California, May 6–8, 1996.

[20] T. D. Garvey and T. F. Lunt. Model-based intrusion detection. In *Proceedings of the 14th National Computer Security Conference*, pages 372–385, Washington, D.C., Oct. 1–4, 1991. National Institute of Standards and Technology/National Computer Security Center.

[21] J. Habra, B. Le Charlier, A. Mounji, and I. Mathieu. ASAX: Software architecture and rule-based language for universal audit trail analysis. In Y. Deswarte et al., editors, *Computer Security – Proceedings of ESORICS 92*, volume 648 of *LNCS*, pages 435–450, Toulouse, France, Nov. 23–25, 1992. Springer-Verlag.

[22] L. T. Heberlein et al. A network security monitor. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, pages 296–304, Oakland, California, May 7–9, 1990.

[23] K. Ilgun. USTAT: A real-time intrusion detection system for UNIX. In *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, pages 16–28, Oakland, California, May 24–26, 1993.

[24] K. Ilgun, R. A. Kemmerer, and P. A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, Mar. 1995.

[25] K. A. Jackson. Intrusion detection system (IDS) product survey. Technical Report LA-UR-99-3883, Los Alamos National Laboratory, Los Alamos, New Mexico, June 25, 1999. Version 2.1.

[26] K. A. Jackson, D. H. DuBois, and C. A. Stallings. An expert system application for network intrusion detection. In *Proceedings of the 14th National Computer Security Conference*, pages 215–225, Washington, D.C., Oct. 1–4, 1991. National Institute of Standards and Technology/National Computer Security Center.

[27] H. S. Javitz and A. Valdes. The SRI IDES statistical anomaly detector. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 316–326, Oakland, California, May 20–22, 1991.

[28] H. S. Javitz, A. Valdes, D. E. Denning, and P. G. Neumann. Analytical techniques development for a statistical intrusion-detection system (SIDS) based on accounting records. Technical report, SRI International, Menlo Park, California, July 1986.

[29] D. L. Kewley and J. F. Bouchard. DARPA information assurance program dynamic defense experiment summary. In *Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, West Point, New York, June 6–7 2000. To appear.

[30] S. Kumar. *Classification and Detection of Computer Intrusions.* PhD thesis, Purdue University, West Lafayette, Indiana, Aug. 1995.

[31] U. Lindqvist. The inquisitive sensor: A tactical tool for system survivability. In *Supplement of the 2001 International Conference on Dependable Systems and Networks*, pages C–14–C–16, Göteborg, Sweden, July 1–4, 2001.

[32] U. Lindqvist and P. A. Porras. Detecting computer and network misuse through the production-based expert system toolset (P-BEST). In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 146–161, Oakland, California, May 9–12, 1999.

[33] U. Lindqvist and P. A. Porras. eXpert-BSM: A host-based intrusion detection solution for Sun Solaris. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001)*, New Orleans, Louisiana, Dec. 10–14, 2001. To appear.

[34] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. Analysis and results of the 1999 DARPA off-line intrusion detection evaluation. In H. Debar, L. Mé, and S. F. Wu, editors, *Recent Advances in Intrusion Detection (RAID 2000)*, volume 1907 of *LNCS*, pages 162–182, Toulouse, France, Oct. 2–4, 2000. Springer-Verlag.

[35] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition (DISCEX)*, volume 2, pages 12–26, Hilton Head, South Carolina, Jan. 25–27 2000.

[36] T. F. Lunt, R. Jagannathan, R. Lee, A. Whitehurst, and S. Listgarten. Knowledge-based intrusion detection. In *Proceedings of the Annual AI Systems in Government Conference*, pages 102–107, Washington, D.C., Mar. 27–31, 1989.

[37] T. F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, P. G. Neumann, H. S. Javitz, and A. Valdes. A real-time intrusion-detection expert system (IDES). Technical Report SRI-CSL-92-05, Computer Science Laboratory, SRI International, Menlo Park, CA 94025-3493, USA, Apr. 1992.

[38] S. McCanne, C. Leres, and V. Jacobson. *libpcap.* Network Research Group, Lawrence Berkeley National Laboratory, Berkeley, California, 1994. Available via anonymous ftp from ftp.ee.lbl.gov.

[39] A. Mounji. *Languages and Tools for Rule-Based Distributed Intrusion Detection.* PhD thesis, Institut d'Informatique, University of Namur, Belgium, Sept. 1997.

[40] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, May/June 1994.

[41] The Netcraft Web server survey, Feb. 2001. *http://www.netcraft.com/survey/.*

[42] P. G. Neumann and P. A. Porras. Experience with EMERALD to date. In *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, California, Apr. 9–12, 1999.

[43] A. One. Smashing the stack for fun and profit. *Phrack Magazine*, 7(49), Nov. 8, 1996. *http://www.fc.net/phrack/files/p49/p49-14*.

[44] J. Picciotto. The design of an effective auditing subsystem. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 13–22, Oakland, California, Apr. 27–29, 1987.

[45] P. A. Porras and R. A. Kemmerer. Penetration state transition analysis: A rule-based intrusion detection approach. In *Proceedings of the Eighth Annual Computer Security Applications Conference*, pages 220–229, San Antonio, Texas, Nov. 30–Dec. 4, 1992.

[46] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353–365, Baltimore, Maryland, Oct. 7–10, 1997. National Institute of Standards and Technology/National Computer Security Center.

[47] P. Proctor. Audit reduction and misuse detection in heterogeneous environments: Framework and application. In *Proceedings of the Tenth Annual Computer Security Applications Conference*, pages 117–125, Orlando, Florida, Dec. 5–9, 1994.

[48] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Calgary, Alberta, Canada, Jan. 1998. *http://www.clark.net/~roesch/idspaper.html*.

[49] RadView Software, Inc., Mar. 2001. *http://www.radview.com/*.

[50] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of LISA '99: 13th Systems Administration Conference*, pages 229–238, Seattle, Washington, Nov. 7–12, 1999.

[51] Sanctum, Inc., Mar. 2001. *http://www.sanctuminc.com/*.

[52] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on TCP. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 208–223, Oakland, California, May 4–7, 1997.

[53] M. M. Sebring, E. Shellhouse, M. E. Hanna, and R. A. Whitehurst. Expert systems in intrusion detection: A case study. In *Proceedings of the 11th National Computer Security Conference*, pages 74–81, Baltimore, Maryland, Oct. 17–20, 1988. National Institute of Standards and Technology/National Computer Security Center.

[54] L. Stein and D. MacEachern. *Writing Apache Modules with Perl and C*. O'Reilly & Associates, 1999.

[55] Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, CA 94043, USA. *SunSHIELD Basic Security Module Guide*, Nov. 1995.

[56] Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303, USA. *SunSHIELD Basic Security Module Guide, Solaris 7*, Oct. 1998. Part No. 805-2635-10.

[57] U.S. Department of Defense. *Trusted Computer System Evaluation Criteria*, Dec. 1985. DoD 5200.28-STD.

[58] H. S. Vaccaro and G. E. Liepins. Detection of anomalous computer session activity. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 280–289, Oakland, California, May 1–3, 1989.

[59] A. Valdes and K. Skinner. Adaptive, model-based monitoring for cyber attack detection. In H. Debar, L. Mé, and S. F. Wu, editors, *Recent Advances in Intrusion Detection (RAID 2000)*, volume 1907 of *LNCS*, pages 80–92, Toulouse, France, Oct. 2–4, 2000. Springer-Verlag.

[60] G. Vigna and R. A. Kemmerer. NetSTAT: A network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71, 1999.

*U.S. GOVERNMENT PRINTING OFFICE: 2002-710-038-10249

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of Information Systems Science and Technology to meet Air Force unique requirements for Information Dominance and its transition to aerospace systems to meet Air Force needs.